NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

OPNET SIMULATION OF SIGNALING SYSTEM NO. 7 (SS7) NETWORK INTERFACES

by

Ow Kong Chung

March 2000

Thesis Advisor:	John McEachen
Second Reader:	Murali Tummala

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2000	**3. REPORT TYPE AND DATES COVERED** Master's Thesis
4. TITLE AND SUBTITLE OPNET Simulation of Signaling System No. 7 (SS7) Network Interfaces		**5. FUNDING NUMBERS**
6. AUTHOR(S) Ow, Kong Chung		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)		**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.		
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

This thesis presents an OPNET model and simulation of the Signaling System No.7 (SS7) network, which is dubbed the world's largest data communications network. The main focus of the study is to model one of its levels, the Message Transfer Part Level 3, in accordance with the ITU-T Recommendation Q.704. An overview of SS7 that includes the evolution and basics of SS7 architecture is provided to familarize the reader with the topic. This includes the protocol stack, signaling points, signaling links and a typical SS7 network structure. This is followed by a more detailed discussion about the functions of the various parts of the protocol, in particular, the functionality of the Message Transfer Parts. The OPNET modeling of the signaling message handling aspect of the Message Transfer Part level 3 is presented. The simulation model presented uses a hierarchical approach, with each level corresponding to the SS7 level it is modeling. Simulation results of different scenarios using varying parameters, such as packet transmission time, packet length, and load sharing, for a typical SS7 network are also presented.

14. SUBJECT TERMS Signaling, Signaling System No.7 kernel, Message Transfer Part, OPNET, Message signaling units			**13. NUMBER OF PAGES** 185
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

OPNET SIMULATION OF SIGNALING SYSTEM NO.7 (SS7) NETWORK INTERFACES

Ow, Kong Chung
Major, Republic of Singapore Air Force
B.Eng., National University of Singapore, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2000

Author: [illegible]
Ow, Kong Chung

Approved by: [illegible]
John McEachen, Thesis Advisor

[illegible]
Murali Tummala, Second Reader

[illegible]
Jeffrey B. Knorr, Chair
Department of Electrical and Computer Engineering

ABSTRACT

This thesis presents an OPNET model and simulation of the Signaling System No.7 (SS7) network, which is dubbed the world's largest data communications network. The main focus of the study is to model one of its levels, the Message Transfer Part Level 3, in accordance with the ITU-T Recommendation Q.704. An overview of SS7 that includes the evolution and basics of SS7 architecture is provided to familarize the reader with the topic. This includes the protocol stack, signaling points, signaling links and a typical SS7 network structure. This is followed by a more detailed discussion about the functions of the various parts of the protocol, in particular, the functionality of the Message Transfer Parts. The OPNET modeling of the signaling message handling aspect of the Message Transfer Part level 3 is presented. The simulation model presented uses a hierarchical approach, with each level corresponding to the SS7 level it is modeling. Simulation results of different scenarios using varying parameters, such as packet transmission time, packet length, and load sharing, for a typical SS7 network are also presented.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AIN	Advanced Intelligent Networks
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
BIB	Backwards indicator bit
BSN	Backwards sequence number
CCITT	International Telegraph and Telephone Consultative Committee
CCIS	Common Channel Interoffice Signaling
CCS	Common channel signaling
DPC	Destination Point Code
FCS	Frame check sequence
FIB	Forward indicator bit
FIFO	First-in-first-out
FISU	Fill-in signal unit
FSN	Forward sequence number
FSM	Finite state machine
HDLC	High Level Data Link Control
HMDC	Message discrimination
HMCT	Message distribution
HMRT	Message routing
ICI	Interface Control Information
LLSC	Link set control
ISDN	Integrated Services Digital Network
ISUP	ISDN user part
LSAC	Signaling link activity control
LSDA	Signaling data link allocation
LSLA	Signaling link activation
LSLD	Signaling link deactivation
LSLR	Signaling link restoration
LSTA	Signaling terminal allocation
LSSU	Link status signal unit
MSU	Message signal unit
MTP	Message Transfer Part

NSP	Network Services Part
OPC	Originating Point Code
OSI	Open System Interconnection
PSTN	Public Switched Telephone Network
SCP	Service Control Point
SCCP	Signaling Connection Control Part
SIO	Service Information Octet
SIF	Signaling Information Field
SK	Simulation Kernel
SLS	Signaling Link Selection
SP	Signaling point
SSP	Service Switching Point
SS6	Signaling System Number 6
SS7	Signaling System Number 7
STP	Signaling transfer point
SU	Signal unit
TCAP	Transaction capability application part
TCP	Transmission Control Protocol
TUP	Telephone user part
UDP	User Datagram Protocol

ACKNOWLEDGEMENT

The author would like to thank A. Prof. McEachen for his help, guidance, and valuable advice during the work of this thesis. His assistance and support in obtaining the various programming software tools, OPNET license, computer notebook (so that I do not have to 'fight' for computer time with my kids!) and technical books is much appreciated. This support was essential in setting the groundwork for this thesis and making this research a really enriching experience.

The author would also like to express his appreciation to Prof. Tummala for generously contributing his time and effort in reviewing this thesis and providing useful comments.

The author would like to express a word of thanks to his fellow student and friend, Lim, C.T., for putting up the paperwork for the procurement of the hardware and books, and for downloading the ITU-T specifications from the web as the procurement of the specifications would have taken months to arrive.

Last but not least, I would to thank my lovely wife for being so understanding and endearing throughout this thesis work as well as our stay in Monterey. Many a times I had to 'neglect' my two wonderful children to work on the thesis but she was always there to 'cover' me.

I. INTRODUCTION

The main objective of this study is to develop a standard SS7 simulation kernel that can be built upon to assist in the analysis of related emerging technologies, such as wireless ATM and Advanced Intelligent Networks (AIN). A primary design objective is to allow for future evolution of the models to incorporate more complex services and procedures with little difficulty. The modeling tool used is MIL 3, Inc's OP*timized* N*etwork* E*ngineering* T*ools* (OPNET) Modeler, version 6.0. OPNET ships with an existing SS7 model; however, this is a non-standard, user-contributed version that does not closely follow the algorithms defined in the International Telecommunication Union (ITU-T) Recommendations for SS7. Additionally, it did not use the standard SS7 nodes. The standard SS7 kernel that was developed in this study is more representative of an actual SS7 network being used in industry. Consequently, it will be easier for future use of the kernel to analyze SS7's impact on operational Department of Defense (DoD) telecommunications projects as well as other related technologies, including transporting SS7 messages over an Internet Protocol (IP) network in IP Telephony.

A. BACKGROUND

In the mid-sixties, the International Telegraph and Telephone Consultative Committee (CCITT) developed a digital signaling standard called Signaling System Number 6 (SS6) that revolutionized the telephone industry. However, as faster and faster data links were developed and used, SS6 was soon considered too slow as it was designed to work on low bit rate channels only. In the mid-eighties, Signaling System Number 7 (SS7), an offspring of SS6, was created to support many more features and applications than its predecessor. Due to its ability to transfer all types of digital information, SS7 is being used to deliver many sophisticated services to users, such as Asynchronous Transfer Mode (ATM), Integrated Services Digital Network (ISDN), and cellular networks. It is now the world's largest data communications network, linking telephone companies, cellular service providers, and long distance carriers together into one large information-sharing network [1]. Hence SS7 has emerged as the standard signaling protocol of all modern telecommunication systems and is deemed to remain the prevalent out-of-band signaling protocol in the telecommunications industry in the years to come.

B. ORGANIZATION

This thesis is organized into seven chapters. Chapter II presents the background of SS7. Its evolution in the mid-eighties, network architecture, nodes and links and protocol are briefly covered in this chapter. Chapter III delves deeper into SS7's Message Transfer Part (MTP) levels, especially MTP level 3, which is the signaling network level, as it is the crux of the simulation carried out in this thesis. This chapter is useful for the reader to appreciate better how the simulation was carried out for this level. Chapter IV provides an overview of the OPNET modeling software that was used for the simulation. This would assist those who are unfamiliar with this powerful software package and also serves as a good refresher for others. Chapter V explains how the OPNET simulation of the SS7 network was carried out. Chapter VI presents the simulation results obtained for a typical SS7 network using the SS7 kernel that was developed. These simulations were carried out using different scenarios or conditions, like varying package sizes, different transmission interval between packages, and with/without load sharing in the links to perform "what-if" studies. Chapter VII concludes this effort and offers some recommendations for future development of this thesis.

Appendices A to P contain the Proto-C source codes written for the simulation of the MTP level 3. Appendix Q provides the source code for the ISUP level that was used for the simulation in Chapter VI.

II. SIGNALING SYSTEM NUMBER 7 (SS7) ARCHITECTURE

A. TYPES OF SIGNALING

The purpose of a signaling system is to transfer control information between switches in a telecommunications network. Early signaling systems used a technique called in-band signaling where the same transmission channel was shared by both signaling and user information, which could be voice or data. This is not an efficient technique as the traffic of the telephone calls and the traffic of the control signals are competing with each other. Newer signaling systems use a separate out-of-band signaling network to carry signaling messages. This type of signaling method is known as Common Channel Signaling (CCS) where a separate (common) channel is used for signaling, as shown in Figure 1. The CCS consists of supervisory functions, addressing, and call information. A CCS channel conveys messages to initiate and terminate calls, check on the status of part of the network, and control the amount of traffic allowed. Thus, call control does not need to compete with voice traffic for the use of the channel.

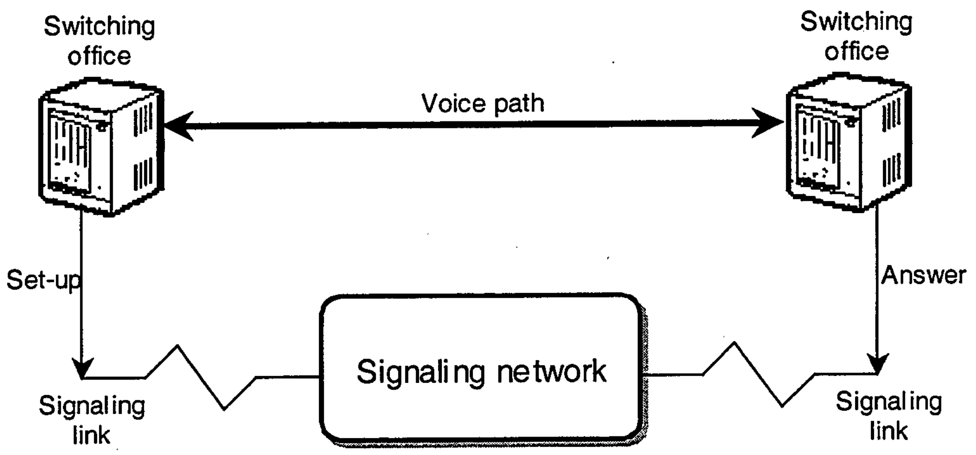


Figure 1. Common Channel Signaling

The advantages of CCS are:

- CCS reduces the call setup time as signaling packets can be efficiently delivered using the signaling link.
- It provides the opportunity to build redundant signaling links between nodes, which improves reliability.
- Signaling can be performed during a conversation.
- It has the ability to look ahead when setting up a connection. Therefore, resources do not have to be reserved until it is determined that a connection can be made, resulting in better utilization of voice trunks.

Thus, the higher reliability of CCS, coupled with increased responsiveness and capabilities, make it an attractive candidate to satisfy telephone carriers' requirements for an improvement to the earlier in-band signaling. It is this method that is used in Integrated Services Digital Network (ISDN) and SS7 today.

B. EVOLUTION OF SS7

In the 1950s and 1960s, Common Channel Signaling Systems were designed for analog networks and were later adapted for digital telephone switches. In 1976, AT&T implemented Common Channel Interoffice Signaling (CCIS) into its toll network. This system is referred to as CCS6 and was based on the International Telegraph and Telephone Consultative Committee (CCITT) Signaling System Number 6 (SS6). Unfortunately, both SS6 and CCS6 were slow and designed to work on low bit rate channels, so they found little application for international signaling. In addition, these architectures were not layered, which made changing the code a complex and expensive task [1].

In 1976, the CCITT began work on a new and powerful digital signaling system. These efforts resulted in the publication of SS7 in 1980, with extensive improvements published in 1984 and again in 1988. Unlike its predecessor, i.e., SS6, which had a monolithic structure, a layered approach to protocol development is used in SS7. This provides for a clear division of functions and an effective means of specifying the complex SS7 protocol. However, the layered approach did not go far enough to conform to the well-known Open System Interconnection (OSI) seven-layered protocol stack [2]. Instead, a four-layer (or “level” in SS7 so as to differentiate it from the OSI layers) protocol stack is used. The reasons were that OSI protocols were concurrently under development at that time, and the initial developments in SS7 were focused on needs specific to telephony signaling. However, as SS7 evolved and matured, the modularity of structure arising from the layered concept, afforded the inclusion of several other applications [3]. The scope of SS7 includes signaling in the public switched telephone network (PSTN), the ISDN, and circuit switched data services. Today, SS7 is also being applied to cellular mobile networks, intelligent networks, and network management.

C. SS7 NETWORK ARCHITECTURE

SS7 is a method of signaling that employs a separate channel for conveying signaling information for a large number of circuit switched channels by means of labeled messages called signal units (SU). From SS7’s point of view, a digital exchange can be considered to be made up of two subsystems – one implementing functions such as call processing, maintenance, and administration, and the other implementing SS7 functions. The part of the exchange that provides SS7 functionality is called a service switching point (SSP). A service switching point has the ability to perform message discrimination as well as to route SS7 messages to another signaling point. The common channel carrying signaling information is called a signaling link. This is illustrated in Figure 2.

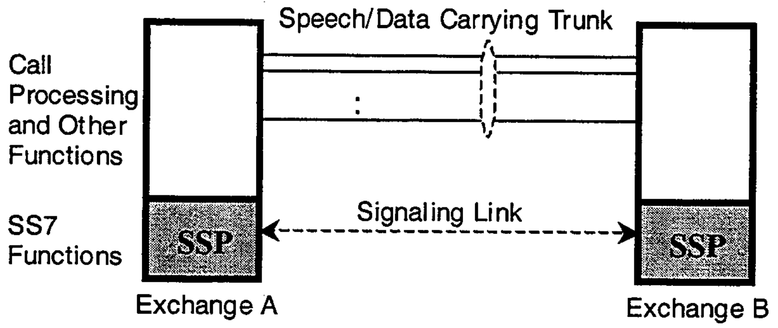

Speech/Data Carrying Trunk
Call Processing and Other Functions
SS7 Functions
SSP
Signaling Link
SSP
Exchange A
Exchange B

Figure 2. SS7 between two adjacent SSPs – in associated mode [3]

SS7 can be used in two possible modes, namely associated mode and quasi-associated mode. With associated signaling, see Figure 2 above, signaling messages pertaining to a particular operation are conveyed over the signaling link that are connected directly between the two signaling points (or nodes). The signal units are transported in both directions on the signaling link. With quasi-associated signaling, the messages are transferred indirectly through at least one tandem point, known as a signaling transfer point (STP). The latter is neither a source nor the destination for the messages. Therefore, an STP merely functions as a transit package switcher for signaling. An STP may be implemented in two ways, either as a stand-alone STP or as an integrated STP (SSP with STP). Figure 3 shows SS7 signaling between two adjacent SSPs in a quasi-associated mode. The mode of signaling to be used – associated, quasi-associated, or a combination of the two – is determined by planning considerations specific to the requirements of individual SS7 networks.

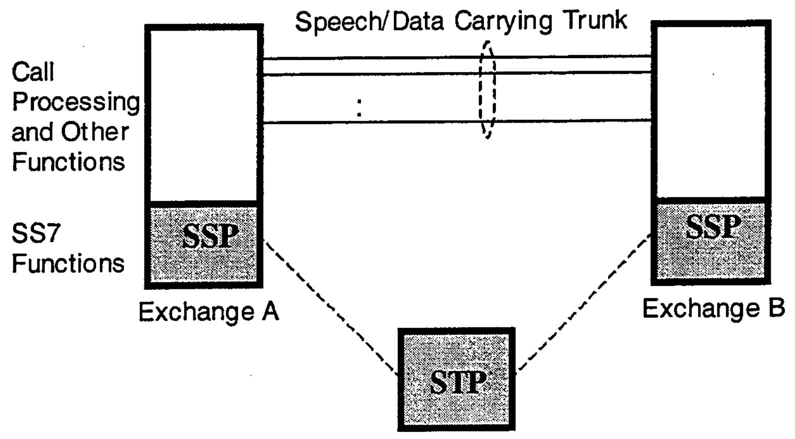

Speech/Data Carrying Trunk
Call Processing and Other Functions
SS7 Functions
SSP
SSP
Exchange A
Exchange B
STP

Figure 3. SS7 between two adjacent SSPs – in quasi-associated mode [3]

D. FUNCTIONS OF THE SS7 NODES

There are three different types of signaling points or nodes in SS7 networks. Figure 4 on the next page shows the three types of signaling points. Major functions of these signaling points are described below.

1. Service Switching Point (SSP)

The Service Switching Point (SSP) is the local exchange in the telephone network. It can be configured as a voice switch, an SS7 switch, or a computer connected to a switch. The SSP provides functionality for communicating with the voice switch via the use of primitives, creating the signaling units at the sending SSP, and translating signal units at the receiving SSP. Therefore, it converts voice signaling into the SS7 signal units, and vice versa. It also supports database access queries for 800/900 numbers to computer systems located centrally to the network. This in fact was the first use of the SS7 network.

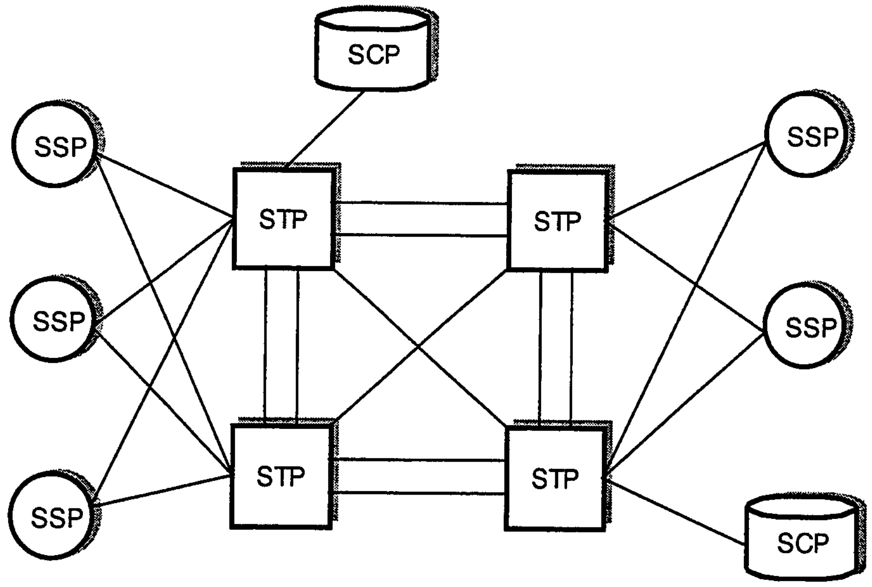


Figure 4. Typical SS7 Network

2. Signaling Transfer Point (STP)

The Signaling Transfer Point (STP) serves as a router in the SS7 network. It relays messages through the network, but it does not originate them. The STP is typically an adjunct to a voice switch, and may or may not stand alone as a separate machine. STPs can be installed as a national STP, an international STP, or a gateway STP. Even though SS7 is an international standard, countries do vary in how some of the features and options are implemented. The gateway STP provides the protocol conversions of the messages that flow from a national standard to the international standard, or some other standard.

STPs also provide screening services, such as security checks on incoming and/or outgoing messages. The STP is able to screen messages to ensure that they conform to the specific network requirements. Other STP functions include subscriber billing and the acquisition and storage of traffic and usage statistics. Hence, the STP is the most versatile of all the SS7 entities, providing a wide array of services to the users of the network.

3. Service Control Point (SCP)

The Service Control Point (SCP) serves as an interface to the telephone company database. These databases contain information on subscribers, routing of special service numbers (such as 800/900 numbers), fraud data, calling card validation and so on. The SCP is usually a computer used as a front end to the database system. The contents of the databases that are made available through SCP vary among SS7 service providers as each service provider has different requirements, so their databases will differ.

E. LINK CLASSIFICATION

All the SS7 signaling points are connected by digital bi-directional signaling links. The information rate can be either 56 kbps or 64 kbps. There are six types of SS7 signaling links as shown in Figure 5 below. The first type, A links or access links, is used to connect SSPs and STPs or SCPs and STPs, usually to an STP pair, which is considered to be its home STP pair.

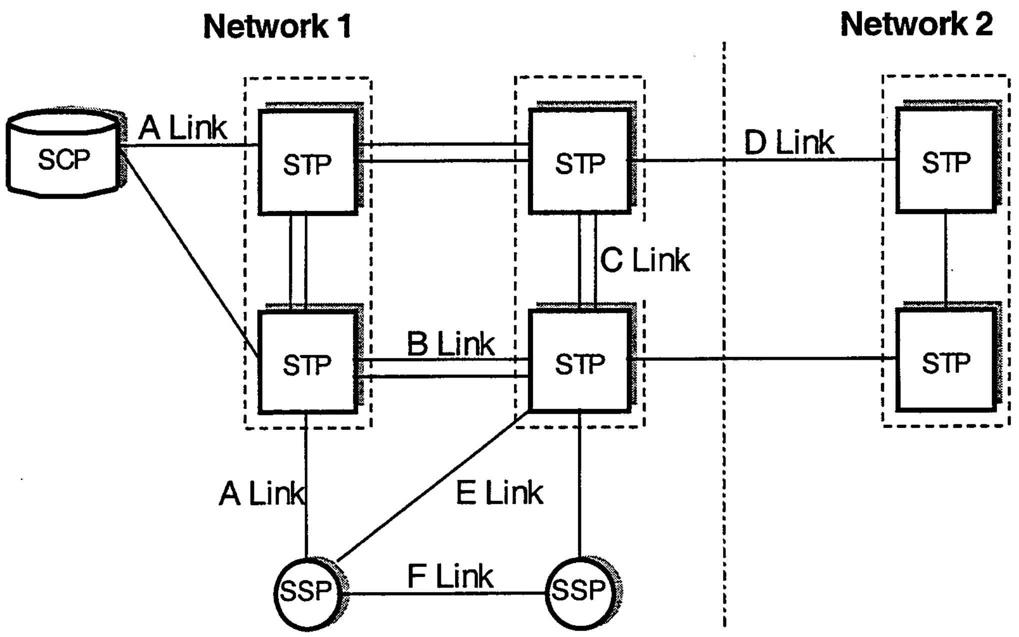


Figure 5. Link classification

B links are also known as bridge links. They are used to connect mated STP pairs to other mated STP pairs. This form of connection is called a quad structure. The connection is on the same hierarchical level, such as between two local STPs.

C links, which are also known as cross links, join STPs together to form mated pairs. The purpose of this type of association is to create redundancy by forming a fully meshed topology among the four STPs. This topology provides 100 percent redundancy. It can be seen that any single point of failure does not bring down the system as the traffic can be diverted around the failure.

Diagonal links or D links are the same as the B links except that the connected STPs belong to different SS7 networks; for example, one in the public switched telephone network and the other in a cellular service network.

The remaining two other types of links are not widely deployed in most SS7 networks. E links, or extended links, provide extra connectivity between an SSP or SCP to mated STP pairs other than its home STP pair. F links, or fully associated links, are those links that connect SCPs or SSPs directly to each other. These links do not connect any STP nodes and are used when the signaling points communicate with each other frequently.

The number of signaling links between two nodes can be up to 128, with the exception of B and D links that are up to 64, though most switch suppliers have limited the numbers to 16 for cost reasons [4].

One of the main advantages of SS7 is its high availability, so in order to support that, SS7 networks are deployed with multiple links among the network signaling points. STPs are configured in pairs and databases/applications are duplicated in different SCPs. When a link error is detected (automatically), the traffic load is shared among the active links.

F. SS7 PROTOCOL

In this section, the SS7 protocol, which is partitioned into four levels, is described. The four levels are:

- Level 1: Signaling Data Link
- Level 2: Signaling Link
- Level 3: Signaling Network
- Level 4: User Parts

Essentially, the first three layers correspond to the first three layers of the OSI Reference model. This component of the SS7 protocol is called the Network Services Part (NSP), and it consists of the Message Transfer Part (MTP) and the Signaling Connection Control Part (SCCP). Figure 6 shows how these relate to each other and to other components of the protocol as well as to the OSI model.

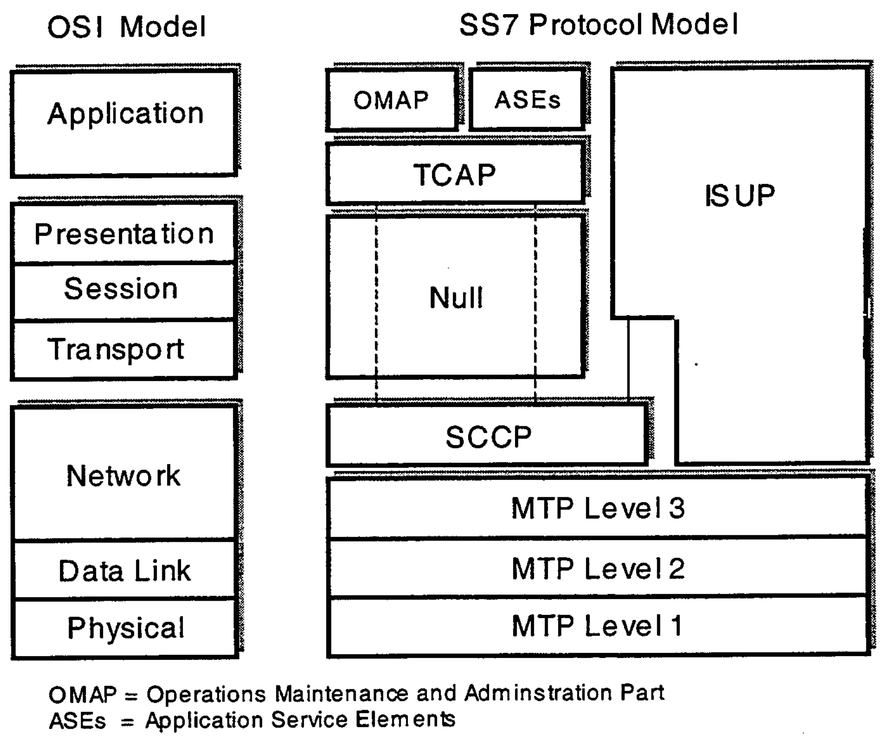


Figure 6. SS7 Protocol Stack vis-à-vis OSI Model [5]

MTP, consisting of levels 1 to 3, provides a connectionless message transfer system that enable signaling information to be reliably and accurately transferred across the network to the desired destination. The reason for its connectionless capability is to avoid the administration and overhead of virtual circuit networks, which was one of the disadvantages of SS7's predecessor, i.e., SS6. MTP also provides the functions necessary for the management of the network. Thus, these three levels focus on the transport of the messages as opposed to the end purpose or application.

The application of signaling to meet specific goals is thus the function of level 4. Many functional blocks in this level representing specific applications use the common message transport facilities offered by the MTP. Since these functional blocks are users of the MTP, they are aptly referred to as user parts. As shown in Figure 6 above, several user parts may exist simultaneously at the user part level. Examples of user parts are the telephone user part (TUP) for signaling in telephony, the ISDN user part (ISUP) for both telephony and the ISDN, and the transaction capability application part (TCAP).

The SCCP is also a user of the MTP. SCCP can be viewed as an enhancement over the MTP transfer capabilities as it provides for both connectionless and connection-oriented network services. It has other users at level 4 too, such as the TCAP and ISUP (which is also a user of the MTP). The splitting of the OSI Network functions into MTP level 3 and SCCP has an advantage in that the higher overhead SCCP services can be used only when required, thus allowing the more efficient MTP to serve the needs of those applications that can use a connectionless message transfer with limited addressing capability.

The SS7 protocol stack is implemented at each SSP in a typical SS7 network. User parts are provided based on the services to be supported. For example, the SS7 network in the ISDN uses the ISUP at the SSPs as illustrated in Figure 7 below. At an STP, only the MTP is present; the user parts are not installed. Ultimately, all communications between the SSPs takes place at the physical level. Functionally, however, the various levels of SS7 at an SSP can be considered to communicate with the corresponding levels of another SSP, which is known as peer-to-peer communication and is shown by the dotted lines in Figure 7.

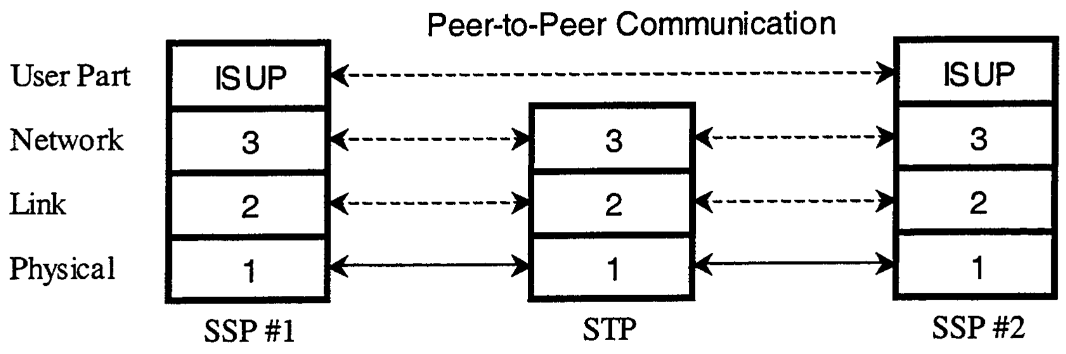


Figure 7. SS7 in an ISDN [3]

THIS PAGE INTENTIONALLY LEFT BLANK

III. MESSAGE TRANSFER PART

The main objective of MTP is to provide reliable transfer and transmission of signaling information across the network, and to take necessary actions in the event of system and network failures to ensure that reliable transfer is maintained. Figure 8 below depicts the functions of MTP levels, and their relationship to one another and to the MTP users.

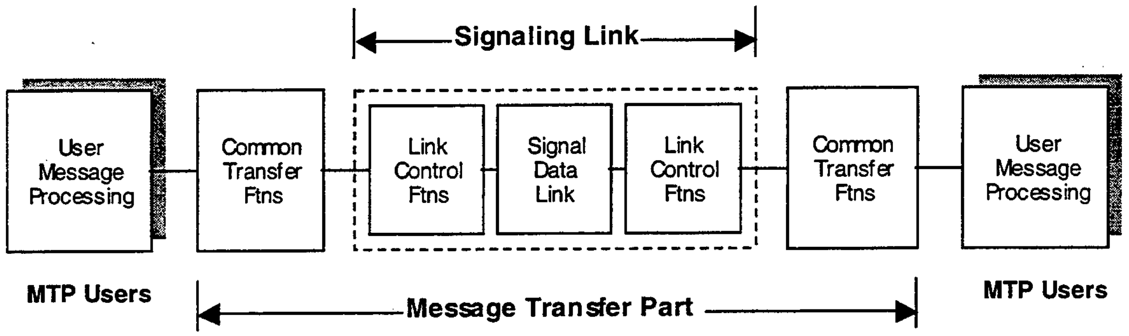


Figure 8. MTP Functional Diagram [5]

This chapter examines these three layers of SS7, which are basic to SS7 protocol and are a mandatory requirement for every implementation. However, the main emphasis will be on MTP level 3. Both MTP levels 1 and 2 are covered in a general discussion.

A. MTP LEVEL 1

A signaling data link consists of two transmission paths operating together in opposite directions at the same data rate and used solely and exclusively for conveying signaling information between two SSPs. The SS7 MTP level 1 signaling data link usually consists of full-duplex, digital transmission channels operating at 56 kbps (which is the standard bit rate for U.S. networks) or 64 kbps (which is CCITT recommended bit rate). Optionally, although seldom used in telecommunication networks nowadays, an analog link can be used with either 3 or 4 kHz spacing, operating at low as 2.4 kbps. For the U.S. network, analog signaling data links are not specified.

SS7 links can operate both on terrestrial or satellite links. However, the latter should be avoided as much as possible due to the large delays inherent in their use. In accordance with clear channel signaling, no other transmission can be transferred with the signaling messages and that is why the MTP links must be solely dedicated to SS7. Any extraneous equipment, such as echo suppressors, must be disabled or removed from the SS7 link.

B. MTP LEVEL 2

This level corresponds to the OSI's data link layer (layer 2). It is quite similar to High Level Data Link Control (HDLC) protocol, as both protocols use flags, error checks, and sending/receiving sequence members. The main difference is that MTP level 2 uses the term signal unit (SU) in place of the more commonly used term - frame. There are three types of SUs in MTP level 2, namely, message signal unit (MSU), link status signal unit (LSSU), and fill-in signal unit (FISU). A high-level view of fields in each of the SUs is shown in Figure 9. The three SUs are described briefly in this section. For a more detailed description of each of the fields in each SU, the reader is referred to [6] or [7].

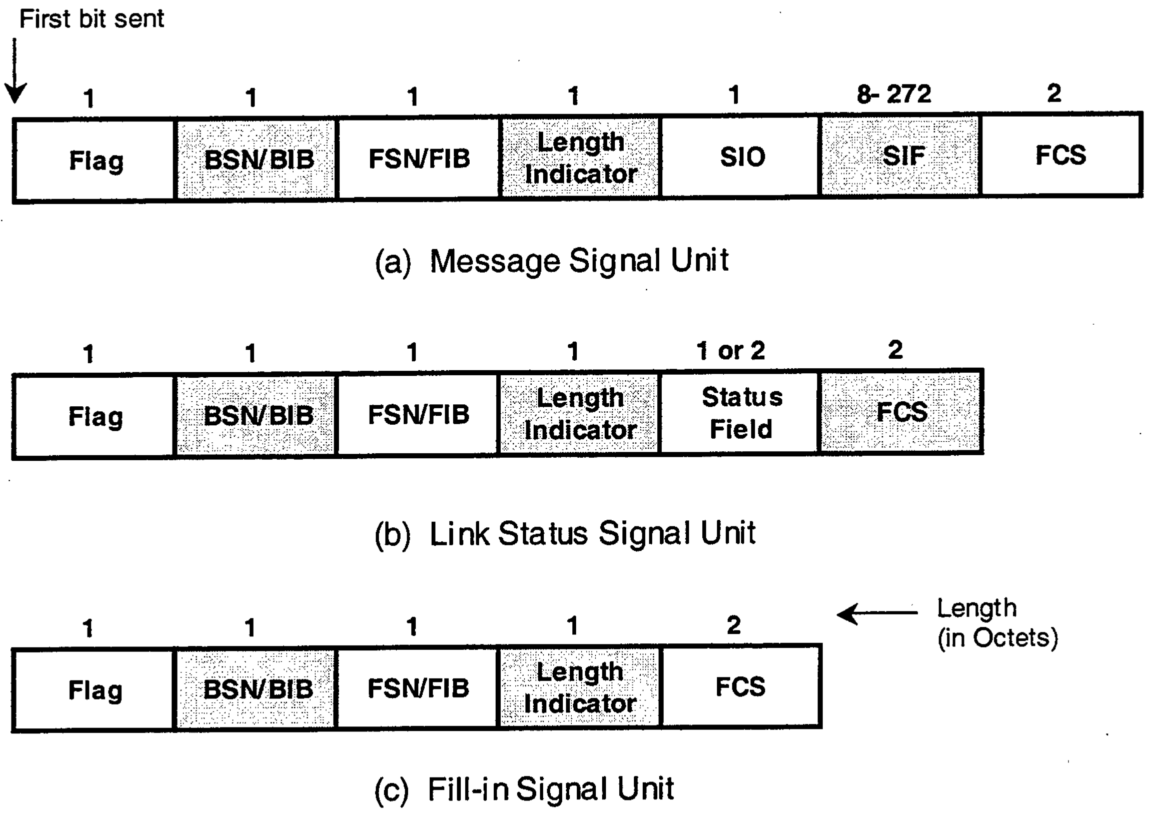


Figure 9. Signal unit formats

1. Message Signal Unit

The signaling message, which is an MTP level 3 message, is carried in the MSUs (see Figure 9a). Messages of other upper layers, such as SCCP, are also carried by the MSUs. Thus, MSUs are the workhorses of the SS7 network. They are the basic envelope within which all addressed signaling information is placed. MSUs are transmitted across the link and if the message is damaged during the transfer, it is then retransmitted.

The functionality of the MSU lies in the actual content of the service information octet (SIO) and the signaling information field (SIF). The SIO, as its name infers, is an 8-bit field that contains three types of information:

1. Four bits are used to indicate the type of information contained in the signaling information field; they are referred to as the service indicator; the values most commonly used in American networks are shown in Figure 10.

Service Indicator	MTP User
0	Signaling network management
1	Signaling network testing and maintenance
3	Signaling connection control part
5	ISDN user part (ISUP)

Figure 10. Common signaling indicator values

2. Two bits are used to indicate whether the message is intended (and coded) for use in a national or international network; they are generally coded with a value of 2 for national network

3. The remaining 2 bits are used to identify a message priority, from 0 to 3, with 3 being the highest priority. The message priorities do not control the order in which messages are transmitted; they are only used in cases of signaling network congestion. In other words, they indicate whether a message has sufficient priority to merit transmission during an instance of congestion or whether it can be discarded en route to a destination.

The SIF in an MSU contains the routing label and signaling information (e.g., SCCP and ISUP message data). For more information about routing labels, refer to the description of MTP Level 3 in subsequent sections.

2. Link Status Signal Unit

The main purpose of the LSSU is to bring up a link, verify that it is operating correctly, and take a link down (see Figure 9b). It is also responsible for link alignment, which entails operations that make sure the traffic on the channel is of the correct length, and that the two SPs are receiving the SUs without any problems. If there are problems, MTP level 2 will liaise with MTP level 3 to take the link out of service, and at the same time make efforts to correct the problems.

3. Fill-in Status Signal Unit

The FISUs perform the function as suggested by their name; they fill up the signaling link until there is a need to send purposeful signaling from MTP level 3 and the upper layers (see Figure 9c). They carry no information payload. Their purpose is to occupy the link at those times when there are no LSSUs or MSUs to send. As FISUs undergo error checking, they facilitate constant link transmission monitoring and the acknowledgement of other SUs using the backwards sequence number (BSN) and backwards indicator bit (BIB).

All three SU types have a set of common fields that are used by this level. They are as follows:

1. The *flags* delimit the SUs. A flag marks the end of one SU and the start of the next.
2. The *checksum* is an 8-bit sum intended to verify that the SU has passed across the link error-free. If the received SU is found to be corrupted, a retransmission is requested.
3. The 6-bit *length indicator* reviews the number of octets between itself and the checksum. It serves both as a check on the integrity of the SU and as a means of discriminating between different types of SUs at this level.
4. The backwards sequence number (BSN), the backwards indicator bit (BIB), the forward sequence number (FSN), and the forward indicator bit (FIB) fields are used to confirm the receipt of SUs and to ensure that they are received in the order in which they were transmitted. They also are used to provide flow control.

MTP level 2 is responsible for seven main link operations [6] and they are briefly outlined here:

1. *Signal unit delimitation.* Uses flags to detect the beginning and ending of the SU.
2. *Signal unit alignment.* Insures the SU length is valid and that any bits between the two legitimate flags are not misinterpreted as flags.
3. *Signal unit error detection.* Uses the FCS field in the SU to detect any bits that were damaged during transmission across the link.
4. *Signal unit error correction.* Provides methods for correcting an error.
5. *Signal unit initial alignment.* Provides services to initialize a link (bring it up) and restore it (upon a failure) by a "proving" process.
6. *Signal unit error monitoring.* Determines if an active link should or should not stay in service and, if the proving process for a new or restored link warrants, makes the link active.
7. *Signal unit flow control.* Controls the flow of SUs between the two signaling points.

For more information on MTP level 2, interested readers are referred to [7] for many of the details.

C. MTP LEVEL 3

There are two basic categories of functionality at this third level: *signaling message handling* and *signaling network management.* Signaling message handling is used for routing messages to the appropriate link and determining if they are addressed to the appropriate link as well as determining if messages are addressed to the received node or are to be forwarded. Signaling network management is used to handle reconfiguration operations in the event when nodes become unavailable. It also controls the flow of traffic in the event of congestion. The breakdown of these functions and their relationship is illustrated in Figure 11.

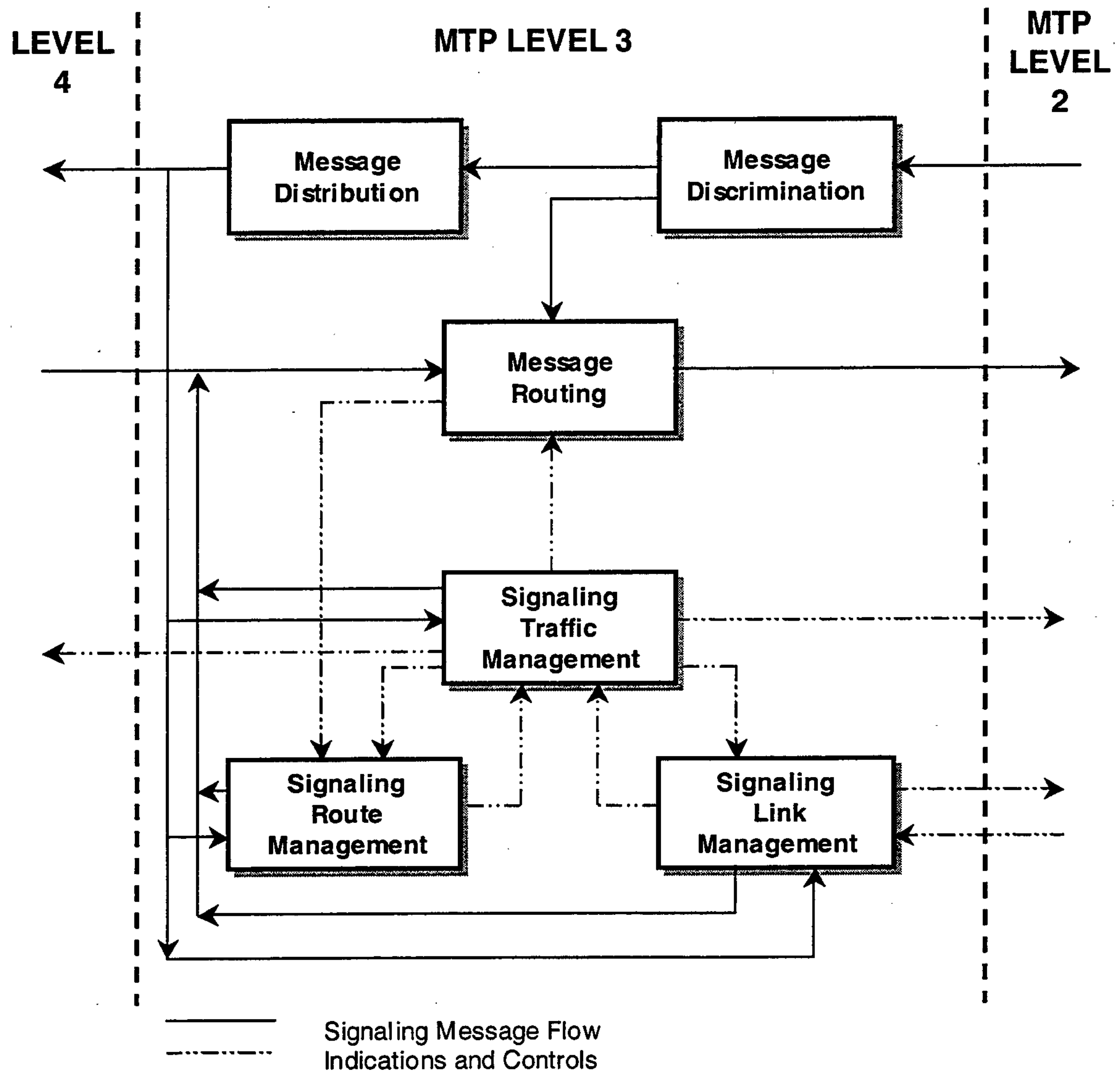


Figure 11. Signaling network functions

1. Signaling Message Handling

Signaling message handling is made up of message discrimination, message routing, and message distribution functions. These functions are performed at each signaling point in a network, and they are based on the 32-bit routing label and SIO portions of the message, as shown in Figure 9a (in the preceding section). Figure 12 below shows the structure of the routing label within the MSU. The label is coded in every MSU message type and is made of the Destination Point Code (DPC), the Originating Point Code (OPC), and the Signaling Link Selection (SLS) fields. The DPC indicates the destination point of the message, whereas the OPC indicates the originating point of the message.

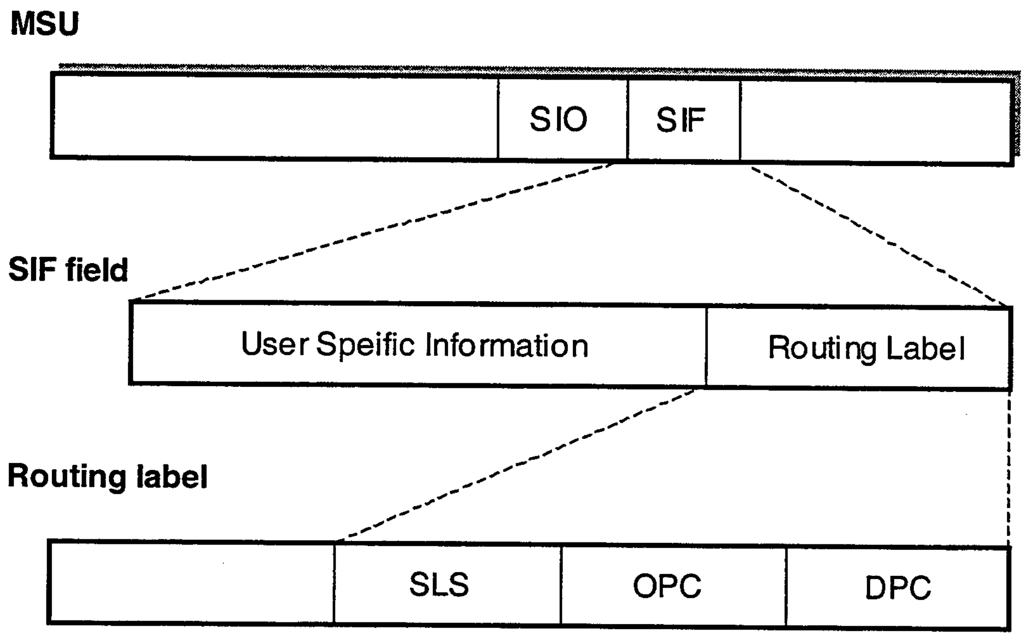


Figure 12. Structure of routing label

In general, there can be more than one signaling link to route a message to a particular DPC. The SLS field is used to select a particular signaling link within a given link set, which is a set of links between two SPs. Load sharing can be done over links in the same link set or over links not belonging to the same link set. The main purpose of load sharing is to keep the load as evenly balanced as possible on the signaling links within a combined link set (which is a load sharing collection of two or more link sets).

a. Message Discrimination

Message discrimination applies to all incoming messages. It uses the routing label of the MSU (see Figure 12 above), specifically the DPC field, to determine if the destination is the receiving node or if the message has to be transferred to another node. If the message is destined for the receiving node, then it is passed to the message distribution function. However, if the message is for another node, then it is passed directly to the message routing function.

b. Message Distribution

Message distribution determines which user part (which is the level 4 function) will receive the message that was handed to it from the message discrimination function. The identification to determine the recipient of the message is done by reading the service indicator octet field. The user can either be a level 4 function, such as ISUP, or level 3 network management.

Hence it can be seen that message discrimination and message distribution functions are simply internal MTP level 3 routers, directing traffic between MTP level 2, MTP level 3, and the upper layers.

c. Message Routing

The message routing function receives messages from level 4 when a newly created message is to be sent, and from the message discrimination function when a message is received but needs to be transferred to another SP. The message routing depends on the DPC (which comprises the network identifier, the cluster identifier, and the member number) located in the routing label (see Figure 12) to determine the destination of the message, and to determine which link the message is to use.

2. Signaling Network Management

The second major part of MTP level 3 is signaling network management functions, which provide reconfiguration of the signaling network in the event of signaling link or SP failures and control the traffic in the event of congestion or blockage.

This way, when a failure occurs, the reconfigurations are quickly carried out so that messages are not lost, duplicated, or become out of sequence, and that the delays do not become too excessive. As depicted in Figure 13 below, the signaling network management function is organized around three complimentary operations: signaling traffic management, signaling route management, and signaling link management. As the details of each of these functions are well documented in the ITU-T standards (Recommendation Q.704) [11], only the key functions will be briefly explained below.

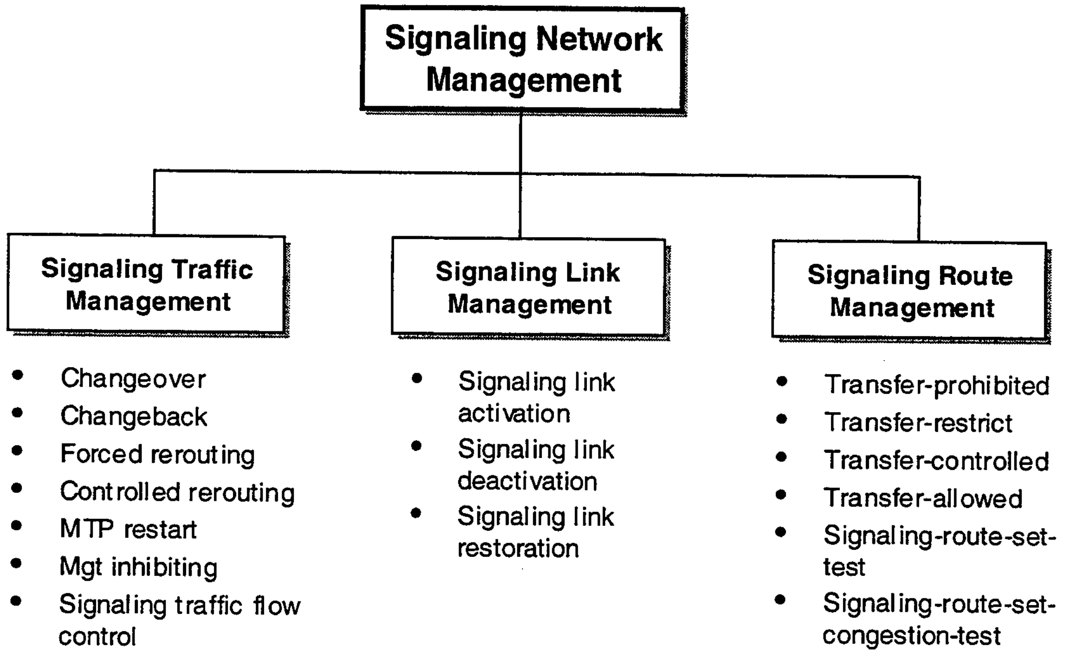

Signaling Network Management
Signaling Traffic Management
Signaling Link Management
Signaling Route Management
Changeover
Changeback
Forced rerouting
Controlled rerouting
MTP restart
Mgt inhibiting
Signaling traffic flow control
Signaling link activation
Signaling link deactivation
Signaling link restoration
Transfer-prohibited
Transfer-restrict
Transfer-controlled
Transfer-allowed
Signaling-route-set-test
Signaling-route-set-congestion-test

Figure 13. Signaling network management

a. Signaling Traffic Management

The signaling traffic management procedures are used to divert signals on one link to one or more alternate links. In addition, it also reduces traffic flow in the event of congestion. In the event of link problems, a *changeover* procedure is used to divert signaling traffic to alternative signaling link(s) and to retrieve for retransmission messages that have not been positively acknowledged. When a previously unavailable signaling link becomes uninhibited, restored, or unblocked, a *changeback* procedure is used to reestablish the signaling traffic on the signaling link made available. When signaling routes (from OPC to DPC) become available or unavailable, *controlled rerouting* and *forced rerouting* procedures are used respectively, to divert the traffic to the restored route or alternative routes. The latter is also used to divert traffic to an alternate but more efficient route when the original route becomes restricted (i.e. less efficient) due to additional transfer points in the path. The *MTP restart* procedure is to allow an MTP at a SP to recover and to bring enough signaling links to the available state to handle all expected traffic. *Management inhibiting* procedure is used to put a link out of service in order to make it unavailable to end user traffic. This procedure is implemented for ongoing maintenance and testing operations. The *signaling traffic flow control* procedure is to reduce the amount of traffic at a source node in the event of failure problems or congestion in the network.

b. Signaling Link Management

The signaling link management function is used to restore failed signaling links, activate idle links, and deactivate aligned links. The set of procedures under this function are *signaling link activation*, which is to activate a signaling link as well as to start the initial alignment procedure for the link, *signaling link deactivation*, which is to deactivate a link assuming there are no active traffic that is running on the link, and *signaling link restoration*, which is to return a previously failed link to service.

c. Signaling Route Management

The signaling route management function is used to disseminate information between SPs in order to block or unblock signaling routes. The following procedures are defined to take care of various situations. The *transfer-prohibited* procedure is performed by an STP to inform its neighbor signaling nodes that they must no longer route certain messages via that STP to a given destination. The *transfer-restricted* procedure is quite similar to the previous procedure with the exception that this procedure informs the adjacent nodes that, if possible, they should no longer route messages to a DPC via that STP. The *transfer-controlled* procedure is also performed by an STP informing its neighbor signaling nodes that they must no longer route certain messages of a given priority or lower via that STP to a given destination. The *transfer-allowed* procedure is used to inform adjacent signaling nodes that routing to a DPC via that STP is now normal. The *signaling-route-set-test* procedure determines if traffic towards a certain destination may be routed through an adjacent STP. Finally, the *signaling-route-set-congestion-test* procedure is used to update congestion status associated with a certain route toward a particular destination.

IV. OVERVIEW OF OPNET

OP*timized* N*etwork* E*ngineering* T*ools* (OPNET) provides a comprehensive development environment supporting the modeling of communication networks and distributed systems. The OPNET environment incorporates tools for all phases of a simulation study, including model design, simulation, data collection, and data analysis. This chapter provides an overview of OPNET's capabilities and structure. It is intended to provide only an introduction to the basic workings of this vast simulation software package. For more detailed or extensive coverage of OPNET, the reader is referred to the OPNET user manuals [8].

A. KEY OPNET FEATURES

OPNET is a vast software package with an extensive set of features designed to support general network modeling and to provide specific support for particular types of network simulation projects [8]. This section gives a brief summary of some of the most important capabilities of OPNET.

1. Object Orientation

Systems specified in OPNET consist of objects, each with configurable sets of attributes. Objects belong to "classes" which provide them with their characteristics in terms of behavior and capability. Definition of new classes are supported in order to address as wide a scope of systems as possible [8].

2. Hierarchical Models

OPNET models are hierarchical, naturally paralleling the layered structure of communications networks and distributed systems. It implements models in three hierarchical levels: the network level, the node level, and the process level.

3. Graphical Specification

Models are generally entered via graphical editors. This provides an intuitive mapping from the modeled system to the OPNET model specification.

4. Flexibility to develop Detailed Custom Models

OPNET provides a flexible, high-level programming language with extensive support for communications and distributed systems. This environment allows for realistic modeling of all communications protocols, algorithms, and transmission technologies.

5. Automatic Generation of Simulations

Model specifications are automatically compiled into efficient, executable, discrete-event simulations implemented in the C programming language. Advanced simulation construction and configuration techniques minimize compilation requirements.

6. Integrated Post-simulation Analysis Tools

Performance evaluation, and trade-off analysis require large volumes of simulation results to be analyzed. OPNET has a sophisticated tool for graphical presentation and processing of simulation output.

B. OPNET EDITORS OR TOOLS

As mentioned in the preceding section, OPNET organizes its model-specification editors in a hierarchical fashion. The Network, Node, and Process editors are the primary development tools. Other tools include Parameter, Probe, Simulation, Analysis and Filter. All these specialized editors address issues at different levels of the hierarchy. This provides an intuitive modeling environment and also allows re-use of lower level models.

At the Network level, the modeler defines the topology of the network, using node and link objects, which are selected from an object palette. The network model must first be defined before the network simulation can be built. The Node editor is used to create node models, which describe the internal flow of data within a network object. Node models consist of modules and connections. Modules are information sources, sinks, and processors. Some modules have pre-defined behavior; processor and queue modules are programmable via their 'process model'. Connections, like packet streams and statistic wires, allow information to flow between modules. The Process editor is used to create process models, which describe the behavioral logic of a module in a node model. It represents models in Proto-C language and uses industry-standard graphical state transition diagrams to define control logic [8]. Proto-C is based on a combination of state transition diagrams, a library of high level commands known as kernel procedures, and the general facilities of the C programming language. This editor is explained in more detail in the following section.

The Parameter editor is used together with process and node model development to define special model structures. The latter allows the modeler to create custom packet formats, Interface Control Information (ICI) formats, probability density functions, link models, modulation functions, and antenna patterns [8]. The Probe editor is a powerful statistics collection editor as it can be used to customize many different types of statistics at user-specified locations.

The Simulation tool provides a convenient environment for configuring and running a single simulation or batch of simulations. Simulation execution typically involves the preparatory step of attaching probes to the points of interest in the model using the Probe editor [8].

The Analysis Tool presents the statistics collected during simulations in the form of two-dimensional graphs or text listings. The information collected by a simulation can be displayed directly, or processed by filters. The Filter editor allows the user to create custom mathematical filters to use in the analysis tool. Filters are built from pre-defined filter components [8].

C. PROCESS EDITOR FEATURES

As the main focus of all OPNET modeling is in defining processes, which help to implement the node model, it is important that the features of the process editor are well understood. So it is worthwhile to devote a section to describe some of the key features available in this editor. A process model is a finite state machine (FSM), which represents a module's logic and behavior. An FSM consists of any number of states that a module may be in and the necessary criteria or condition for changing states, as shown in Figure 14.

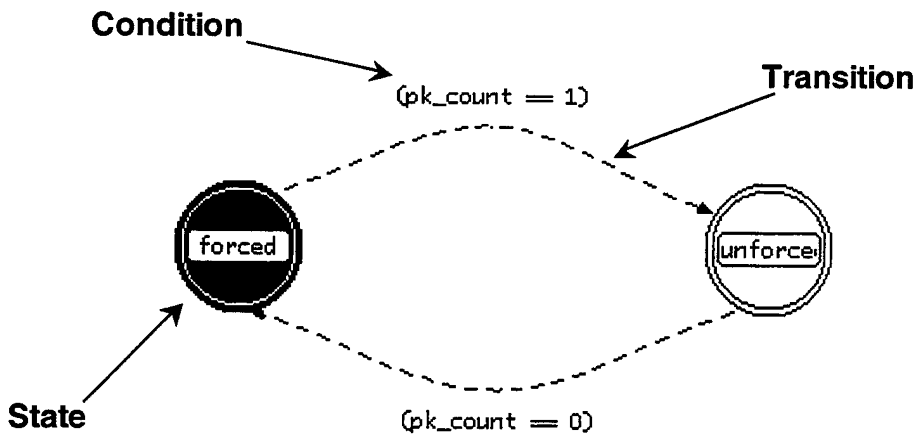
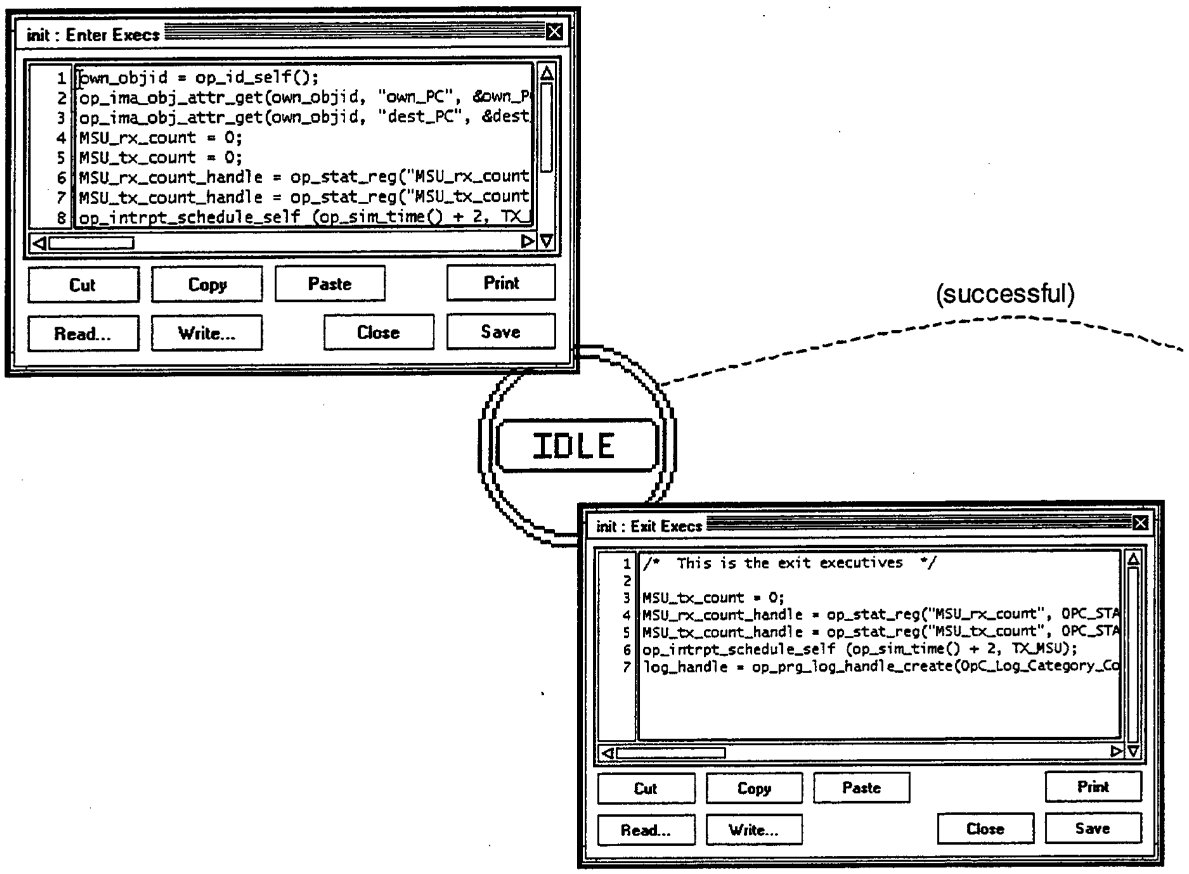


Figure 14. Finite State Machine

OPNET allows the modeler or user to attach fragments of C/C++ language code to each part of the FSM. This code, augmented by OPNET-specific functions, is called Proto-C. The three primary places to use Proto-C codes are:

- *Enter Executives*: Code executed when the module moves into a state,
- *Exit Executives*: Code executed when the module leaves a state, and
- *Transition Executives*: Code executed in response to a given event.

This is illustrated in Figure 15 in the next page.



```
own_objid = op_id_self();
op_ima_obj_attr_get(own_objid, "own_PC", &own_P
op_ima_obj_attr_get(own_objid, "dest_PC", &dest
MSU_rx_count = 0;
MSU_tx_count = 0;
MSU_rx_count_handle = op_stat_reg("MSU_rx_count
MSU_tx_count_handle = op_stat_reg("MSU_tx_count
op_intrpt_schedule_self (op_sim_time() + 2, TX_
```

```
/*  This is the exit executives  */

MSU_tx_count = 0;
MSU_rx_count_handle = op_stat_reg("MSU_rx_count", OPC_STA
MSU_tx_count_handle = op_stat_reg("MSU_tx_count", OPC_STA
op_intrpt_schedule_self (op_sim_time() + 2, TX_MSU);
log_handle = op_prg_log_handle_create(OpC_Log_Category_Co
```

Figure 15. Proto-C codes embedded in FSM

1. Forced and Unforced States

The reader may have noticed the different color shading of the states in the FSM as shown in Figure 14 above. This is because states in OPNET are either *forced* or *unforced.* The main difference between the two types of states is their execution timing.

A forced state, which is green if in color or opaque if in black and white, allows no interruptions (from the Simulation Kernel – see next section) during its execution. It controls the simulation until it has completed all its required actions. So in a forced state, the process will first invoke the enter executives, followed by the exit executives. It then evaluates all condition statements and if exactly one condition statement evaluates to true, the transition is traversed to the next state, as shown in Figure 16.

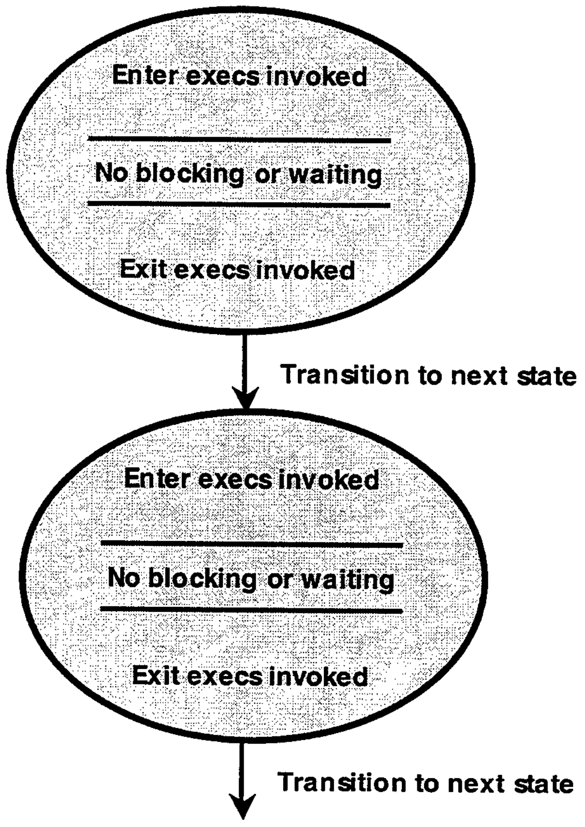


Figure 16. Forced States [9]

On the other hand, an unforced state, which is red if in color or transparent (or white) if in black and white, transfers control back to the Simulation Kernel (SK) once the enter executives of the state have been executed. Thus in an unforced state, the process will first invoke the enter executives. It then places a marker at the middle of the state and it releases control to the SK and becomes idle. When it is next invoked, it will then resume at the marker and processes the exit executives, see Figure 17.

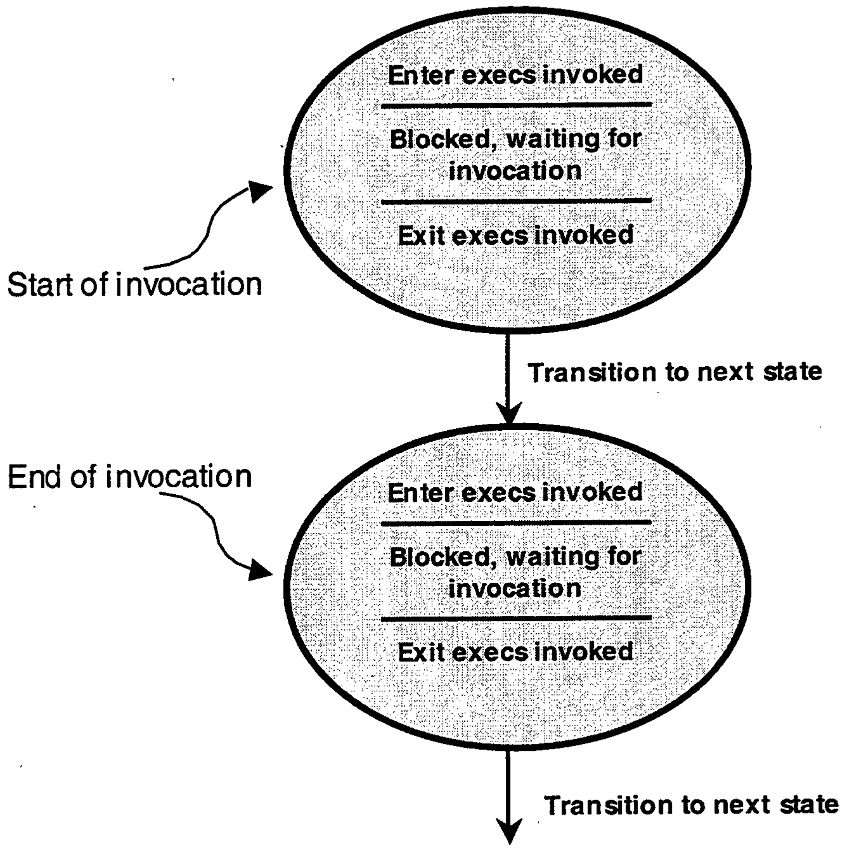


Figure 17. Unforced States [9]

2. Process level action buttons

The process level action buttons, as depicted in Figure 18 below, are meant for writing the Proto-C code, which defines the behavior of the states in the process level. Using each of the SV, TV, HB, FB, DB, and TB action buttons will invoke the OPNET Proto-C editor for the user to key in the necessary Proto-C source code.

Variables in OPNET are declared in two places. The SV action button defines one of them - *state variables*. State variables are those variables that are persistent and will retain their value from invocation to invocation. The other variables, *temporary variables*, as their name suggest, are non-persistent or temporary variables that exist only during the current invocation of the process. Such variables are entered in the temporary variables block by choosing the TV action button.

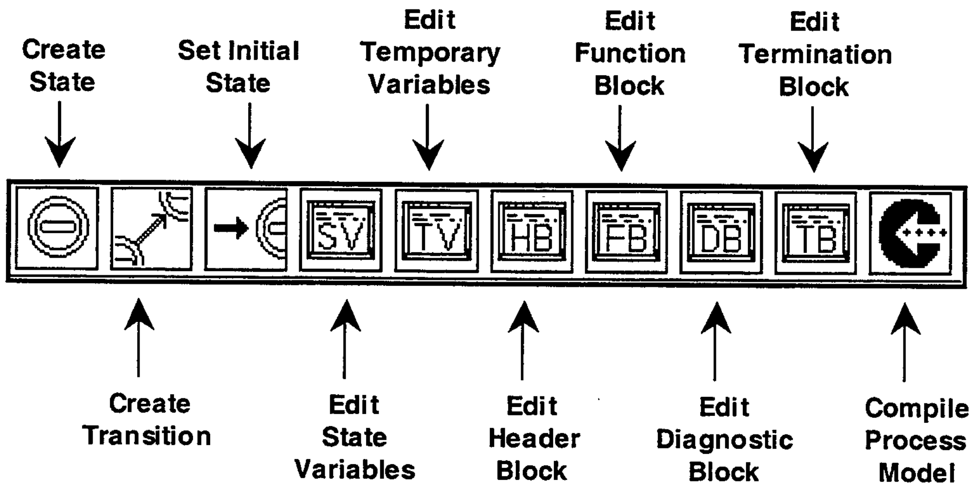


Figure 18. Process level action buttons

The use of macros often replaces the more complicated expressions in transition conditions and executives. This saves space and simplifies the task of interpreting an FSM diagram. Macros are specified by the user with the C/C++ preprocessor *#define* statement, and appear in the process model *header block*, using the HB action button. The header block may also contain *#include* statements, struct definitions, and global variable declarations.

The FB action button defines the *function block*, which is where the modeler will define the various functions used in the process level. Functions are normally invoked from one of the states, but they can also be invoked from another function. The use of functions is a good programming tool as the number of codes in the states can be kept small and thus making the FSM diagram more readable, as the bulk of the processing is off-load to the various functions. As both the *diagnostic block* and the *termination block* are rarely used, the reader can refer to [8] for detailed explanations for these two blocks.

All process models must be compiled before they can be used in a simulation and this can be done by clicking the Compile Process Model action button. Once a model is compiled, its object code is then made available to a processor module's process model attribute.

D. OPNET SIMULATIONS

This section will describe how a simulation works in OPNET, how the Simulation Kernel (SK) manages the event list and also how a process handles an interrupt.

OPNET simulations are based on a discrete-event modeling approach. A simulation moves the model through a sequence of states in order to represent an actual system. A variable, known as *simulation time*, is maintained to model the progress of time as the model state changes. The points in time at which the state of the model can change are called *events*. In other words, an event is a request for a particular activity to occur at a certain time. Some common events might include the receipt of a packet, indication of completion or partial completion of a task by a resource, and expiration of a timer. Simulation time will not change during an event, but only between events. It also does not necessarily advance by the same amounts at each event, but rather it jumps to the next event's time. An OPNET simulation maintains a global event list. Events are scheduled on the list in time order. The first event on the list is the head. When an event is completed, it is removed from the event list.

An entity, known as the Simulation Kernel (SK), uses a proprietary and efficient algorithm to manage the event list. The kernel receives requests from modules and inserts new events on the event list. It then extracts the events from this list and executes them in time sequence. Hence the event list will grow or shrink as new events are inserted or executed during the period of the simulation. The simulation is completed once the event list becomes empty. Figure 19 below depicts the working of the event list.

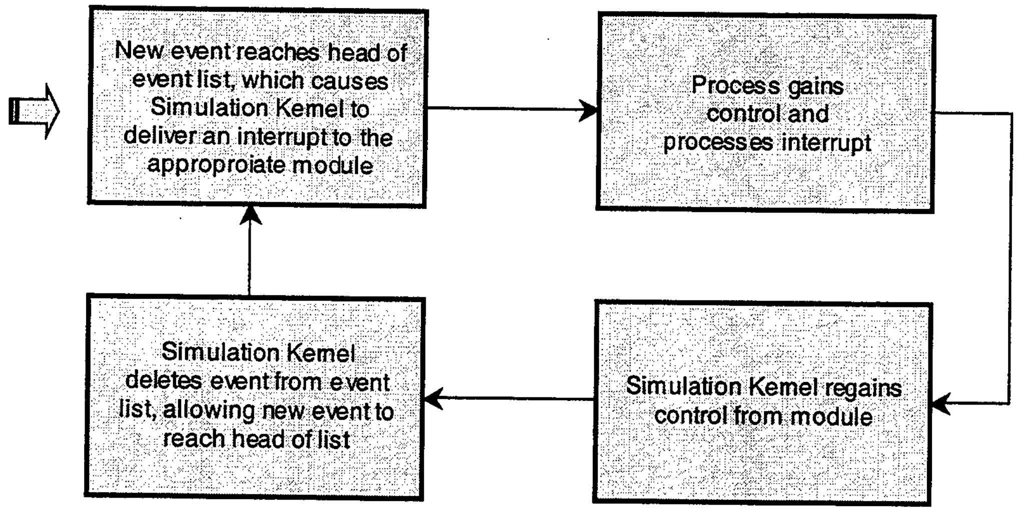

New event reaches head of event list, which causes Simulation Kernel to deliver an interrupt to the approproiate module
Process gains control and processes interrupt
Simulation Kernel regains control from module
Simulation Kernel deletes event from event list, allowing new event to reach head of list

Figure 19. How the event list works. [9]

Processes are normally invoked into execution when the simulation time reaches the time the particular process is scheduled to be invoked or the kernel alerts the process by sending an *interrupt* to the process. An interrupt occurs when an event reaches the earliest position in the event list and becomes executed. The interrupt is sent to the event's destination object and has most of the same attributes as the event. There are various types of interrupts and the more common ones are:

- BEGSIM, which occurs at the start of a simulation, i.e. at time = 0.0,
- ENDSIM, which initiates an interrupt at the end of a simulation,
- STRM, which initiates an interrupt when a packet arrives at a module,
- STAT, which initiates an interrupt when a statistic on a statwire changes, and
- SELF, which initiates an interrupt for a future time at the initiating process.

BEGSIM is a special type of interrupt that occurs at the start of the simulation, before any other type of interrupt. As this is the first interrupt, the process begins execution at the initial state (INIT), marked with an arrow, as shown in the example of Figure 20.

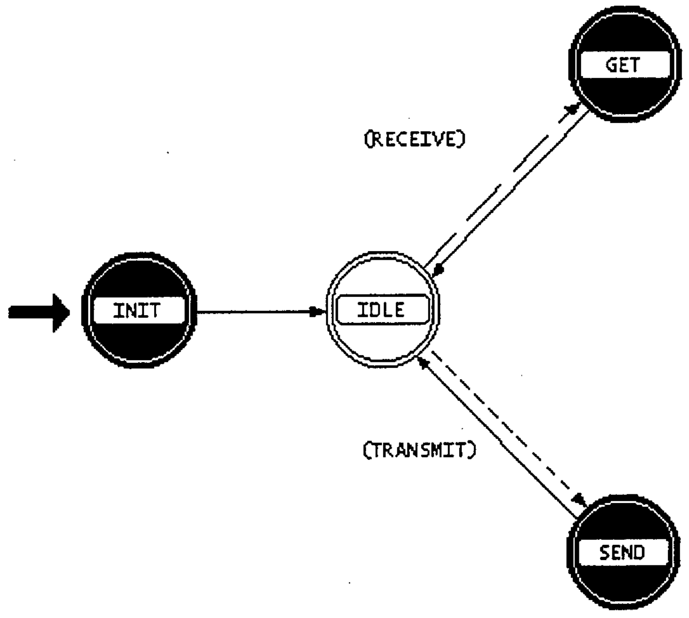


Figure 20. A simple process model example

Since the INIT state is being entered, the enter executives are executed. Being a forced state, the process will immediately invoke and complete the exit executives, evaluate all condition statements and transit to the next state, which is the IDLE state in this example. All these are done without any elapsed time. The IDLE state, being an unforced state, will return the control of the simulation to the SK. Upon regaining control, the SK will check the event list to determine the next scheduled event and the time that it is to occur. At the appropriate time, the SK will invoke the event via an interrupt. Figure 21 is a flow diagram showing how a process handles an interrupt, other than the initial interrupt, BEGSIM.

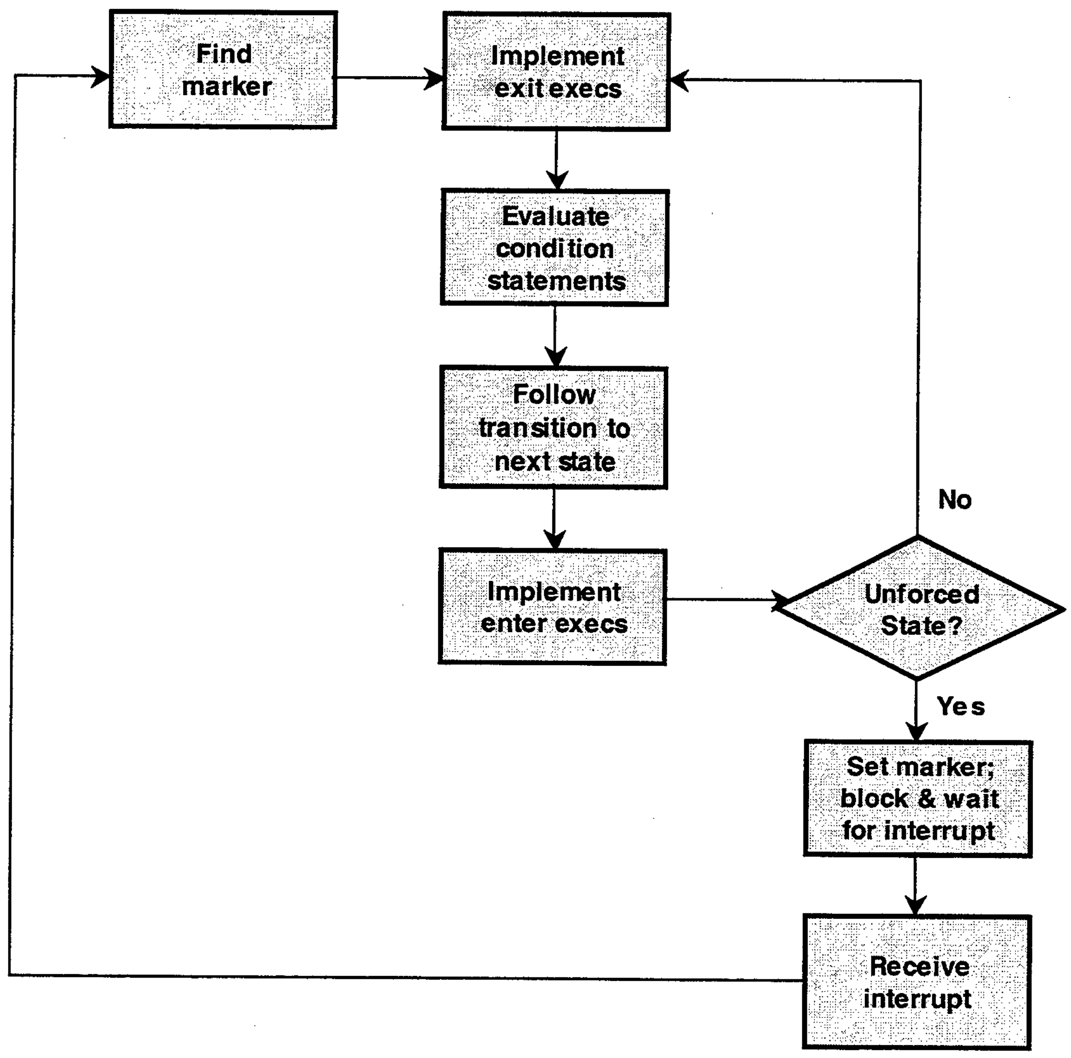

Find marker
Implement exit execs
Evaluate condition statements
Follow transition to next state
Implement enter execs
Unforced State?
No
Yes
Set marker; block & wait for interrupt
Receive interrupt

Figure 21. Flow diagram showing the handling of an interrupt

V. MODELING THE SS7 KERNEL

This chapter describes how the SS7 kernel is developed using the powerful OPNET modeling software package. The three basic MTP levels and an ISDN user part were developed in OPNET to complete the SS7 kernel. However, the main emphasis of this chapter will be on the modeling of MTP level 3. The reader is referred to [9] for a comprehensive description on the modeling of MTP level 2. The ISDN user part was included in the modeling as an application layer so that a complete SS7 network can be simulated in OPNET.

A. MODELING THE SS7 LEVELS

1. MTP Level 1 and 2

The signaling data link, or MTP level 1, is essentially two transmission paths operating in opposite directions but at the same data rate. It is used for conveying signaling information between two SSPs. This level can be easily modeled in OPNET by selecting a pair of point-to-point receiver and transmitter in the Node Editor, see Figure 22.

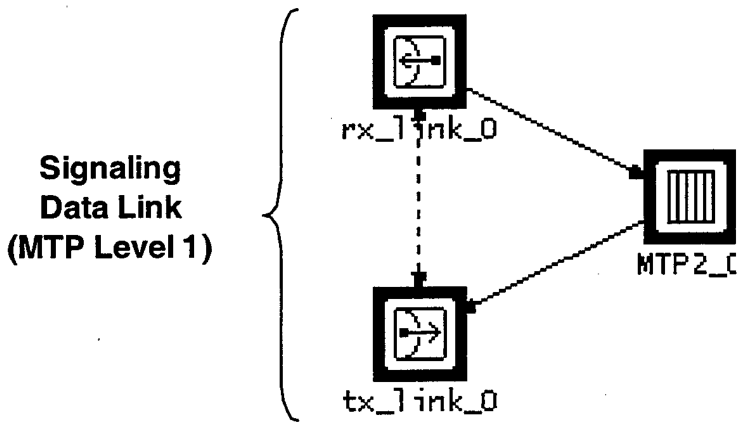


Figure 22 Modeling the signaling data link (MTP Level 1)

As mentioned at the start of this chapter, the modeling of MTP level 2 (with basic error correction) will not be covered in details here. The basic error correction method is a non-compelled method in which correction is performed by retransmission. The modeling follows very closely the algorithms of each of the functional blocks as defined in Figure 7 of the ITU-T Recommendation Q703 (Jul 96) [7]. The functional blocks are:

- Basic Reception Control
- Basic Transmission Control
- Link State Control (LSC)
- Initial Alignment Control (IAC)
- Processor Outage Control (POC)
- Alignment Error Rate Monitor (AERM)
- Signal Unit Error Rate Monitor (SUERM)
- Congestion Control (CC)

The node model of MTP level 2 is illustrated in Figure 23 and Figure 24 shows how all these functional blocks are modeled at the process level of OPNET.

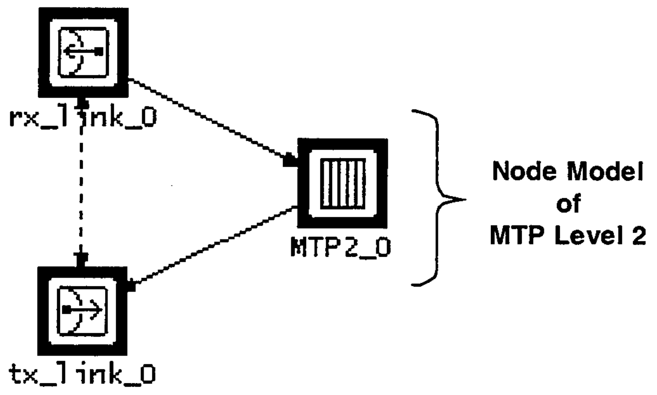


Figure 23. Node Model of MTP Level 2

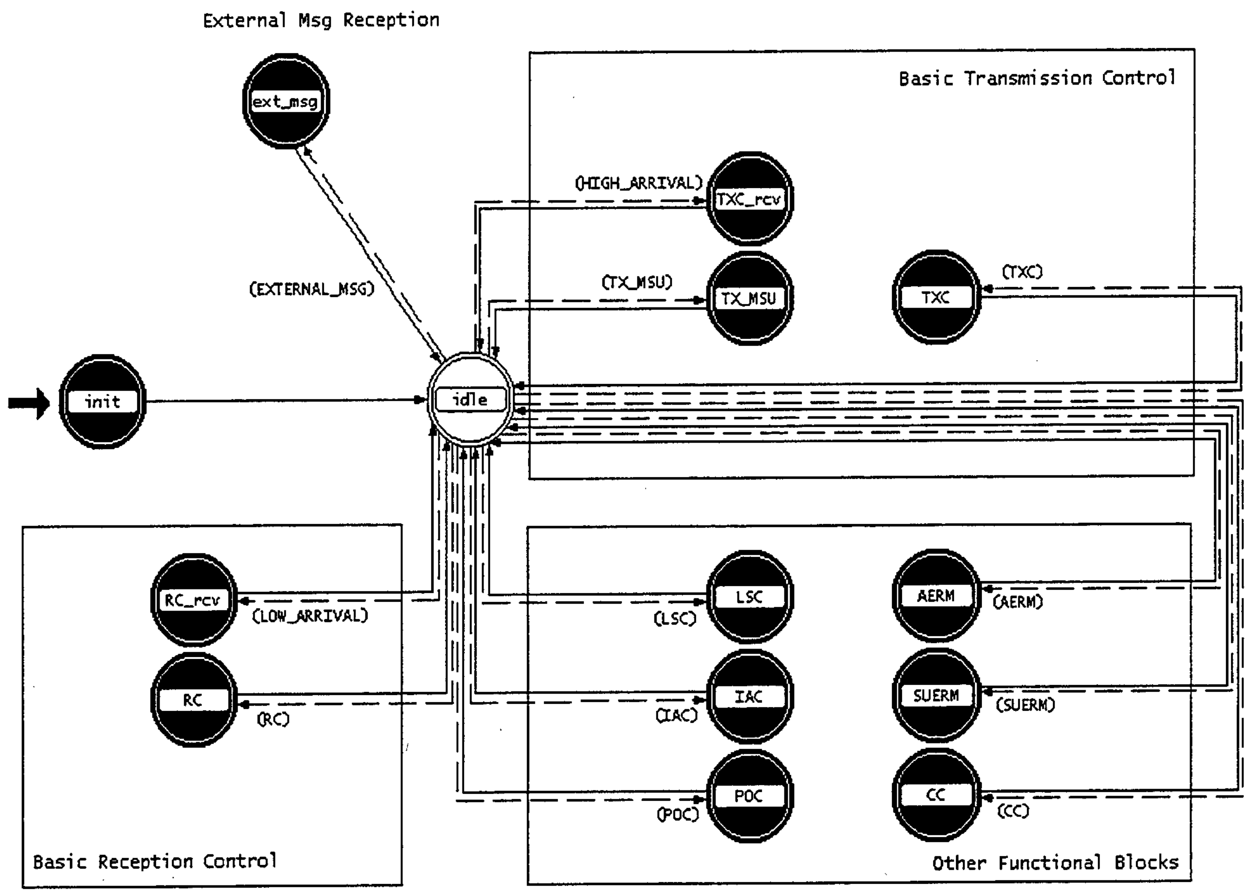


Figure 24. Process model of MTP Level 2 [9]

2. MTP Level 3

The reader may recall from Chapter III that at the MTP level 3, there are two basic categories of functionality, namely, signaling message handling and signaling network management. The modeling of this level follows the algorithms given in ITU-T Recommendation Q704 (Jul 96) [11]. As this recommendation did not provide the algorithms of the functions of the signaling network management but rather only guidelines as to how these functions should be implemented, the latter functions were thus not implemented in the model presented in this thesis. This conforms with the objective to model a basic SS7 kernel that closely follows the algorithms as specified in the ITU-T Recommendations. Besides, leaving out the implementations of the signaling network management functions does not in any way affect the performance simulation of the SS7 network. However, for completeness purpose, the functional blocks of the signaling network management have been included. The latter blocks will simply destroy any message packets routed to them. These blocks are marked with an 'x' in front of the name of the function block, as shown in Figures 25b, 25c, and 25d.

The process model of MTP level 3 is shown in Figure 25. As the picture is too large to be clearly shown in a page, it has been divided into four portions – Figure 25a to 25d. Figure 25a illustrates how the signaling message handling is modeled in OPNET. At the start of the simulation, the *INIT* state is invoked and its enter executives are executed. The INIT state takes care of all the simulation initializations, such as getting the assigned values for all the model attributes, starting the message processing for all the functional blocks, initializing the ICIs, timers and states of the functional blocks, loading the node and link tables (which are discussed in subsequent sections), creating the link route and user availability tables, and determining the message routing (HMRT) destination list. The process will immediately invoke and complete the exit executives as the INIT state is a forced state. It will then evaluate all condition statements, however this state has only one departing transition, which evaluates to true. Consequently, the process transits to the next state, which is the *IDLE* state. This state, being an unforced state, will return the control of the simulation to the SK. Upon regaining control, the SK will check the event list to determine the next scheduled event and the time that it is to occur. At the appropriate time, the SK will invoke the event via an interrupt. It takes approximately 8.53 seconds to complete the initializations in the simulation.

The *receive* state is created to route the arriving packet or message depending if the packet came from the upper layer, in which case it will store the message in the HMRT, or from MTP level 2, in which case it will store the message in the message discrimination (HMDC) input buffer. Refer back to Figure 11 of Chapter III. Note that an infinitely large input buffer is created for each functional block. The messages here refer to only SS7 signaling units. Messages other than SS7 signaling units are handled by the external message processing state, *ext_msg*. All such messages intended for this are received here and are recognized as an interrupt with type "OPC_INTRPT_REMOTE".

The ‘code’ field of the interrupt is then retrieved to determine the message and if necessary, the ‘ici’ field of the interrupt can also be read (if more data were also sent). All received messages are then queued into the message buffer of the respective functional block for processing.

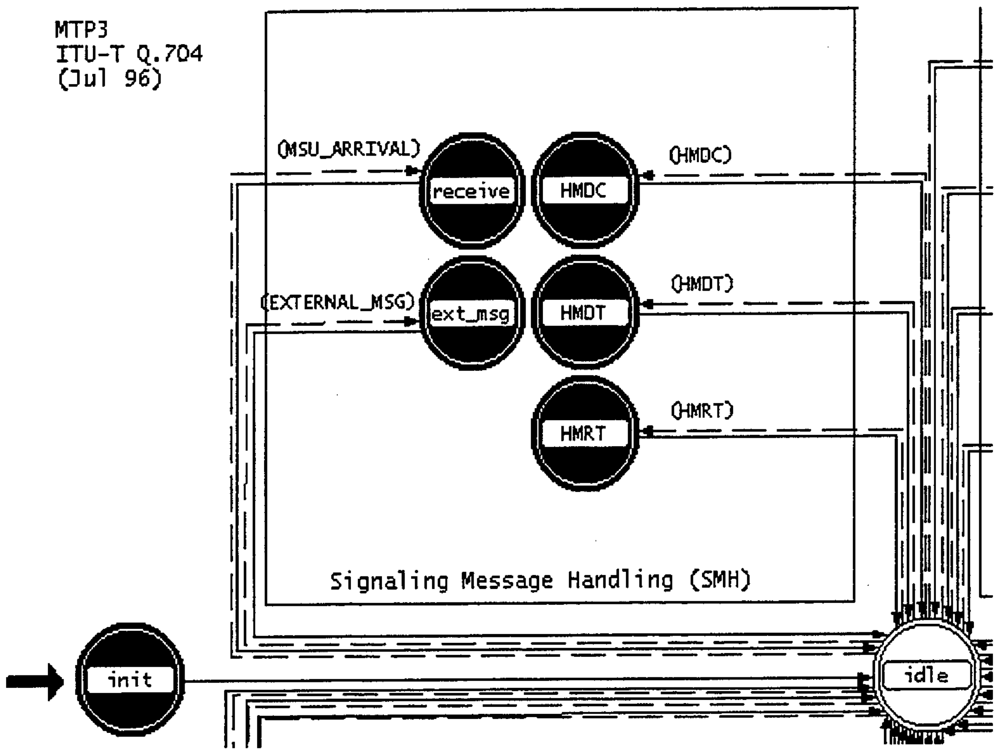


Figure 25a. Process Model of MTP Level 3 - Signaling message handling block

As noted in Chapter III, the signaling message handling function is subdivided into three other functional blocks and each of these are modeled as a state in the OPNET simulation. The message discrimination state, *HMDC*, defines all HMDC processing. The reader may recall from Chapter III that this includes handling of all incoming internal messages and discrimination of MSUs from level 2. This is done by examining the destination point code of the incoming message. Messages are processed in a first-in-first-out (FIFO) manner. Each message is processed according to the algorithm as defined in Figure 24 of [11].

The *HMDT* state defines all the message distribution processing. This essentially is to determine which user part will receive the message that HMDT has just received from HMDC. This is accomplished by examining the service indicator field to see if it is a level 3 network management message or meant for a level 4 user. Like the HMDC state, the messages in HMDT are also processed in a FIFO manner. The processing algorithm follows those as specified in Figure 25 of [11].

Similarly, the message routing state, *HMRT*, does the message routing based on the destination point code in the routing label of any incoming messages available in the input buffer. These messages can come from HMDC, Network Management (i.e. Signaling Traffic Management, Signaling Route Management, and Signaling Link Management), Network Testing and Maintenance, and other level 4 users. Note that at the end of the source codes for all the states, the packet or message must be destroyed using the op_pk_destroy() command so as to free up the memory space.

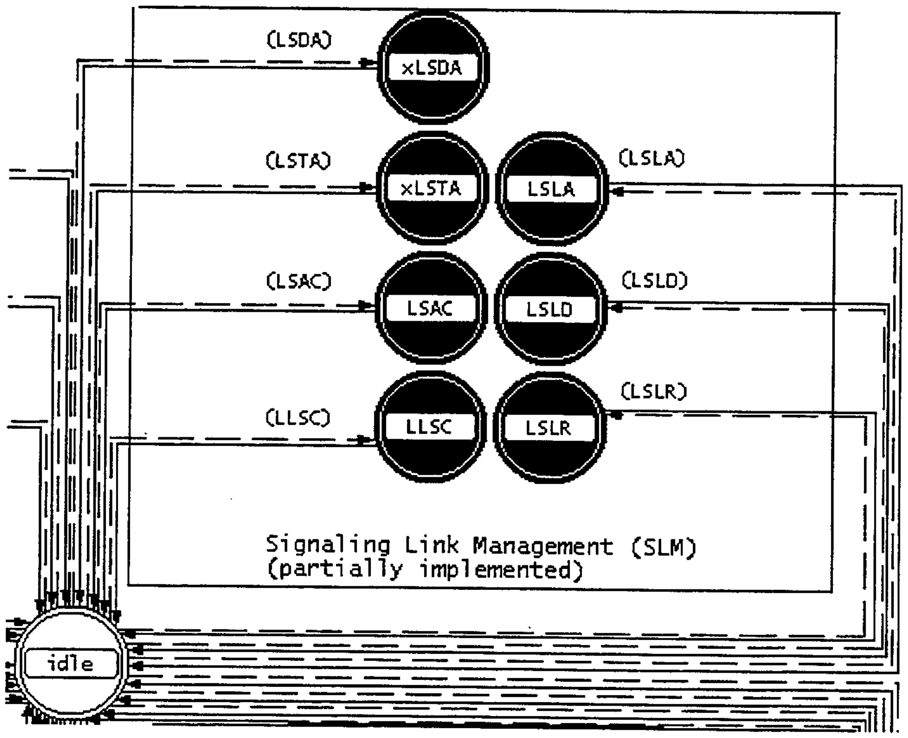


Figure 25b. Process Model of MTP Level 3 - Signaling link management block

The signaling link management function is also subdivided into smaller functional specification blocks as specified in Figure 35 of [11]. Using the algorithms defined in Figures 36 to 42 of [11], some of these functional blocks are implemented as states as illustrated in Figure 25b on the preceding page. As stated before, those states marked with an ‘x’ in front of the state name are not implemented, for example, signaling data link allocation (LSDA) and signaling terminal allocation (LSTA). The implemented functional blocks are link set control (LLSC), signaling link activity control (LSAC), signaling link activation (LSLA), signaling link restoration (LSLR), and signaling link deactivation (LSLD). All these functional blocks interact with one another to restore failed signaling links, activate idle links, and deactivate aligned links.

Both the signaling route management and the signaling traffic management functions were not implemented, however each of their smaller functional specification blocks is still modeled into states (for completeness) and are shown in Figures 25c and 25d respectively.

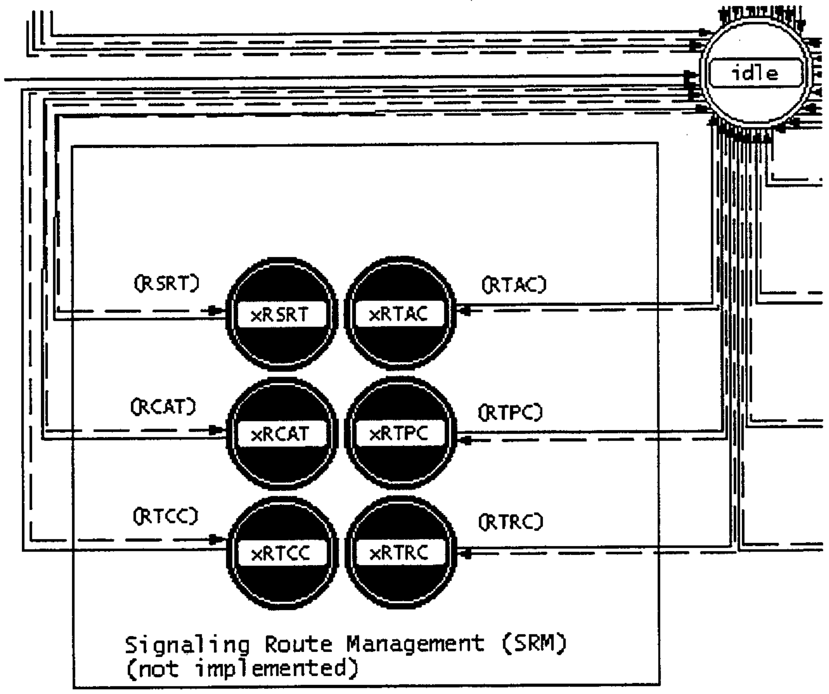


Figure 25c. Process Model of MTP Level 3 - Signaling route management block

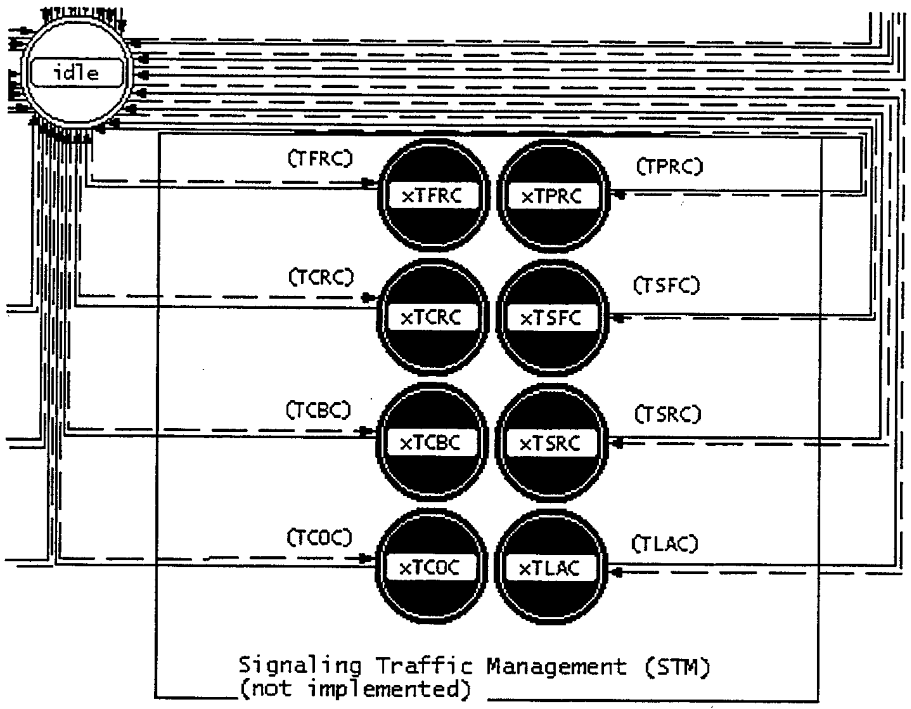


Figure 25d. Process Model of MTP Level 3 - Signaling traffic management block

The node model of MTP level 3 is illustrated in Figure 26.

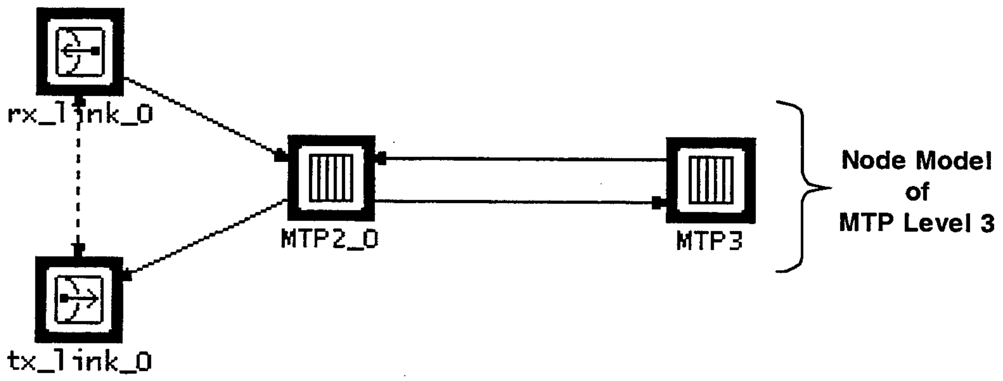


Figure 26. Node Model of MTP Level 3

3. Level 4 – ISUP

In order to simulate a SS7 network, an application layer is needed. For this level, the ISDN user part (ISUP) was modeled with a *GET* state that counts the number of messages received and a *SEND* state that creates and transmits messages as well as keeps track of the number of messages sent out. The process and node models are as shown in Figures 27 and 28 respectively.

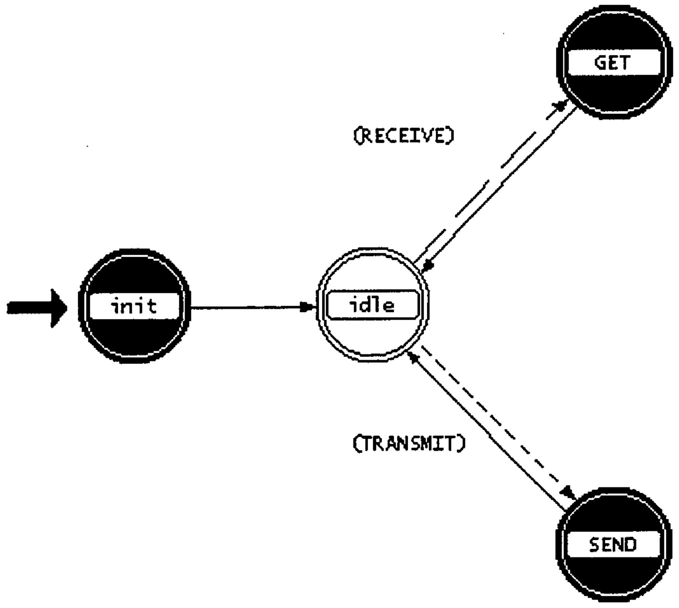


Figure 27. Process model of the ISUP (level 4)

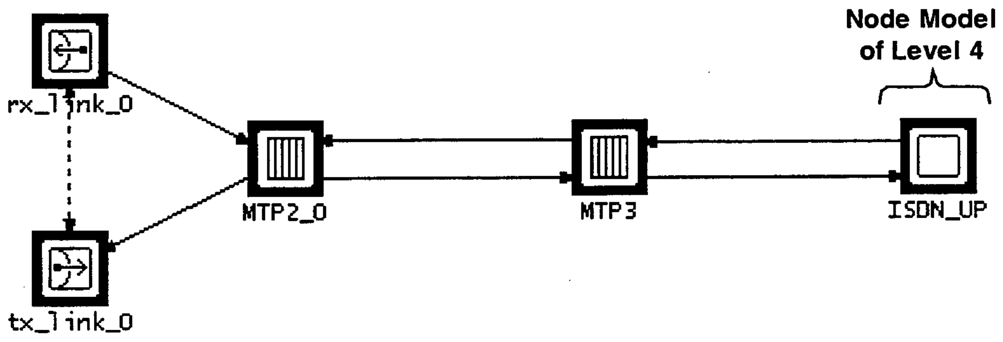
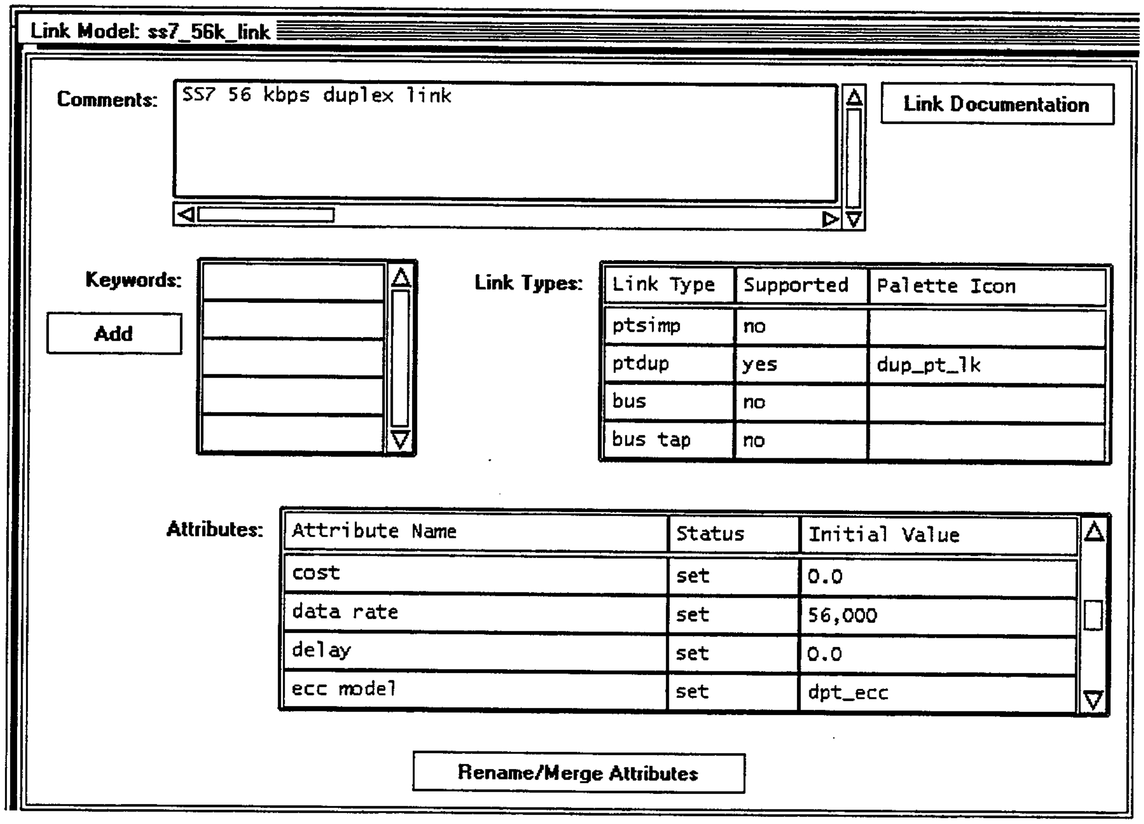


Figure 28. Node model of the ISUP (level 4)

4. Link model

The definition of reusable link models that are used to create link objects in the Project Editor is done using the Link Model Editor. Each link object's attribute interfaces, presentation, and behavior, are determined by the link model that it relies upon. One can view the link model attributes as the 'parameters' of the model, allowing the modeler to control its behavior in predefined ways. As SS7 uses signaling data links that are full-duplex and operating at 56 kbps or 64 bps, these two types of links were created for our SS7 model. Figure 29 below shows how the 56 kbps data link is created using the Link Editor. Note from the figure that the link type is a point-to-point duplex type and the data rate is set to 56,000 bps.



Comments: SS7 56 kbps duplex link

Link Types:

Link Type	Supported	Palette Icon
ptsimp	no	
ptdup	yes	dup_pt_lk
bus	no	
bus tap	no	

Attributes:

Attribute Name	Status	Initial Value
cost	set	0.0
data rate	set	56,000
delay	set	0.0
ecc model	set	dpt_ecc

Figure 29. Creating a link object using the Link Editor

5. Network model

Now that the node, link, and process models have been constructed to define the topology of each of the SS7 levels and links, it is appropriate to model an SS7 network at the network level. The network model will specify the entire SS7 system to be simulated. To do that, the object palette for the SS7 simulation must first be defined.

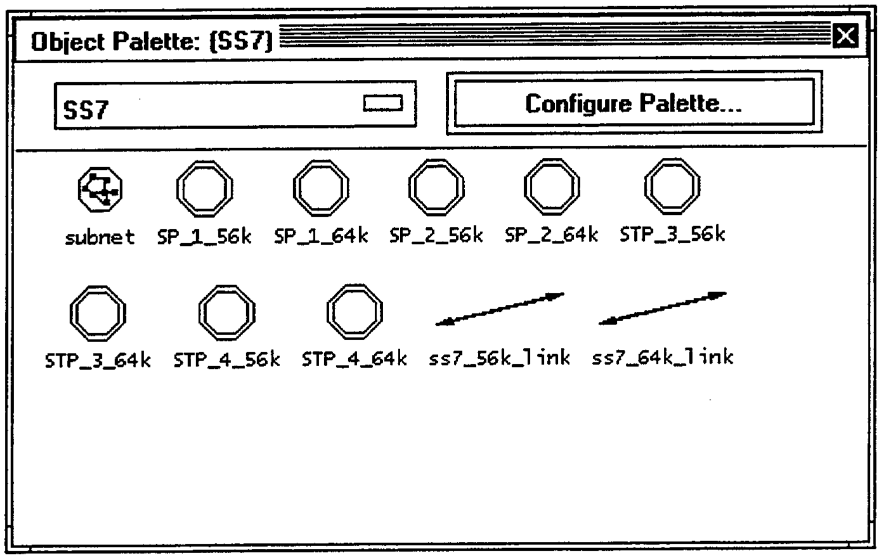


Figure 30. Object palette for our SS7 model

Figure 30 above shows the objects that were constructed for our SS7 model. In this SS7 palette, we have defined an SSP/SCP object with one link (i.e., SP_1_56k) or two links (i.e., SP_2_56k) connected to it as well as a STP with either three links (i.e., STP_3_56k) or four links (i.e., STP_4_56k) connected to it. Similar to the links, both 56 kbps and 64 kbps SS7 nodes were constructed. Using these objects, an SS7 network model was created as shown in Figure 31. Note that the user can choose to change the symbol of any object. The symbol for the STP nodes used in Figure 31 is that of a file server, which is more representative of an actual STP. Also the modeler needs to assign an arbitrary user_id number to each of the SS7 nodes.

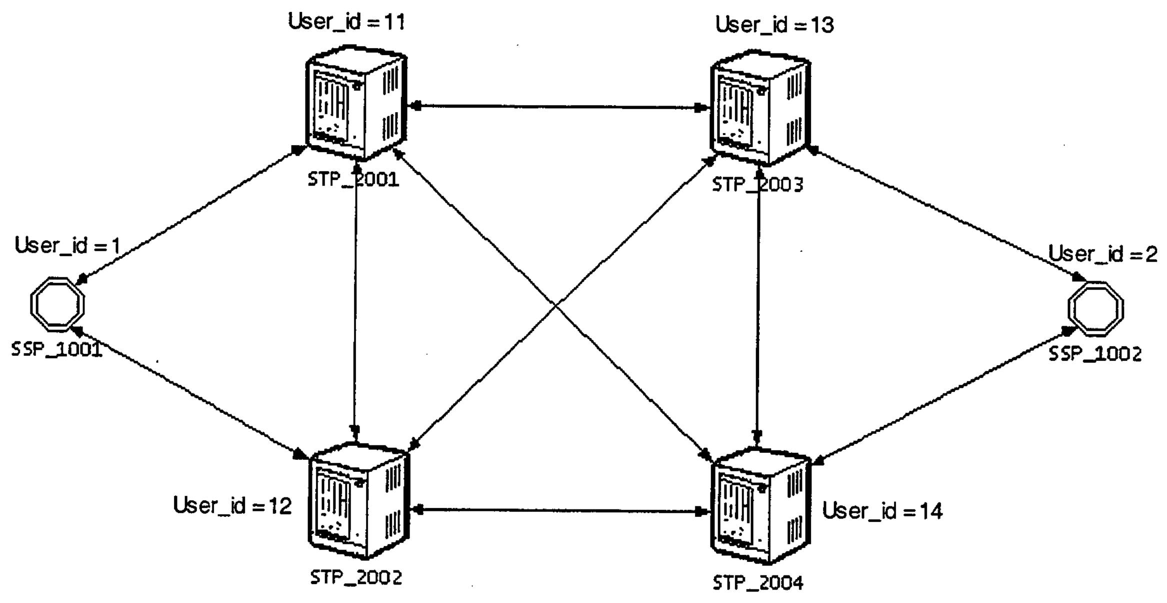


Figure 31. Network model of an SS7 network

Before starting the simulation, all the nodes (SSP and STP) and the links must be configured with identification data, such as point codes and user id. Two templates, namely node.gdf and link.gdf, have been created (under Notepad) for the modeler to key in the data specific to the SS7 network he wants to model. In both the templates, no comments are allowed and each data is separated by a comma but with no spacing in between. An important point is that when saving the files in Notepad, both files must have an extension of '.gdf'. Notepad will automatically insert its own extension of '.txt' onto the filename despite the fact that '.gdf' extension has already been inserted. Hence the '.txt' extension must be deleted for the files to run correctly. For the node.gdf template, the data required for each node are: the subnet user id, user id, point code, and type (where 0 represents an SP and 1 represents an STP). The data in each row represents those of a node in the network.

```
100,1,1001,0
100,2,1002,0
100,11,2001,1
100,12,2002,1
100,13,2003,1
100,14,2004,1
```

Figure 32. Node data for all the nodes of the SS7 network of Figure 31

Note that in Figure 32, an arbitrary number of 100 was chosen as the subnet user id.

For the link.gdf template, the data required for each link is: the node point code (which is the point code of the node the link originates from), stream number (which is the equivalent to the port number the link and is obtained by looking at the extended attributes of the link), and SP_connected number (which is the number of the node that the link connects to). The data in each row represents those of a link in the network. Figure 33 shows the link data for our network.

```
1001,10,2001
1001,11,2002
1002,10,2004
1002,11,2003
2001,10,1001
2001,11,2003
2001,12,2002
2001,13,2004
2002,10,2001
2002,11,2004
2002,12,2003
2002,13,1001
2003,10,2001
2003,11,2002
2003,12,1002
2003,13,2004
2004,10,2002
2004,11,2003
2004,12,1002
2004,13,2001
```

Figure 33. Link data for all the links in the SS7 network of Figure 31.

Now that both the node and link templates have been completed, the modeler can begin the simulation by choosing the required simulation run time and placing appropriate statistic probes to collect the required data he/she wants to examine. The next chapter will present some simulation results obtained for a typical SS7 network that was described in Figure 31 above.

VI. SIMULATION RESULTS

In the previous chapter, we saw how a simulation model can be built in OPNET using the SS7 kernel that was developed in this thesis. In this chapter, we will present the simulation results obtained from the simple SS7 network shown in the last chapter (see Figure 31). These simulations were carried out using different scenarios or conditions, like varying package sizes, different transmission interval between packages, and with/without load sharing in the links. These various conditions are incorporated in the Proto-C code of the ISDN user part, found in Appendix Q.

A. SIMULATION SETUP

In this section, we will describe the scenarios used for our simulations and any underlying assumptions made for the simulations. Figure 31 from the previous chapter is duplicated here in Figure 34 for easy reference. For simplicity, it is assumed that the

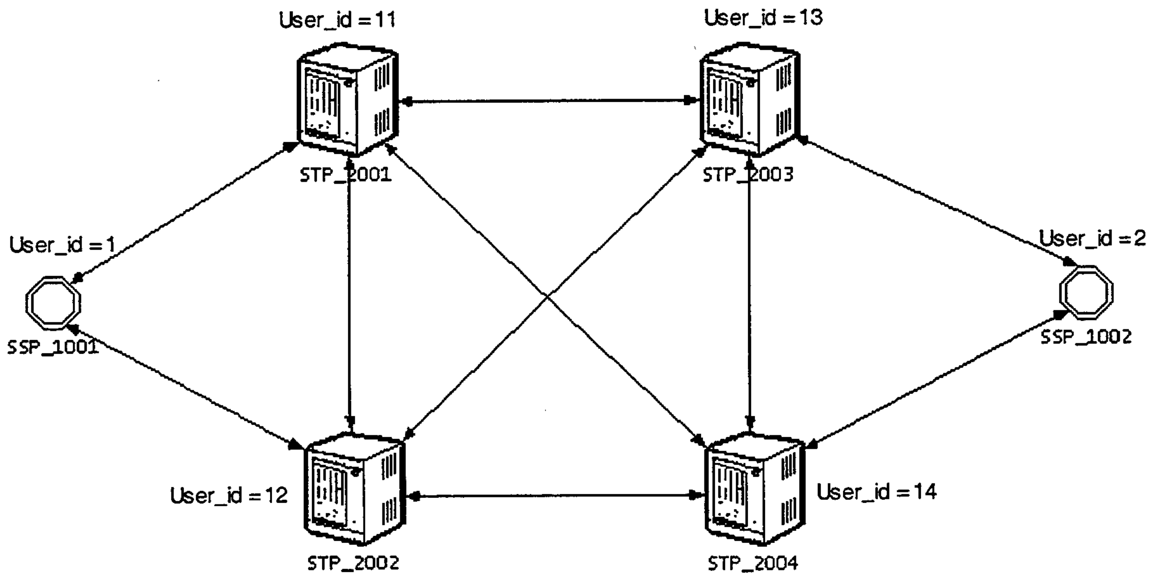


Figure 34. Network model of an SS7 network

propagation delays in each of the links are negligible. This is possible if the SS7 nodes of the network shown in Figure 34 are not too far away from one another. Of course, if one is to model a huge network that spans across thousands of miles, the link propagation delays can be easily incorporated into the simulation. All messages are assumed to be processed in a FIFO manner and all the buffers used in the SS7 simulation are of infinite length.

As stated earlier, an ISUP layer was written to incorporate the scenarios for our simulation as well as to collect the result, i.e., the total average (steady-state) delay of a packet that was sent from one SSP to another. As the reader may recall, ISUP is simply a circuit-related protocol that is used for establishing circuit connections and maintaining the connections throughout a call. The first parameter that may be varied in our simulation is the packet length of the ISUP message (encapsulated on the SS7 MSUs). The most common packet-length distribution is the exponential distribution. As there are many ISUP messages that can be transmitted by the 'send' state in the ISUP layer, it is assumed that the average length of the message is 40 bytes long [1]. Within this 'send' state, we have the following definitions:

```
remainding_packet_len = op_dist_exponential(184);
op_pk_fd_set(pkptr, FD_INDEX_8TH, OPC_FIELD_TYPE_INTEGER, 0,
              remainding_packet_len);
```

The op_dist_exponential is an OPNET kernel procedure that generates an exponential random value with a mean of 184 (in this case). The number 184 comes about as the SS7 MSUs fields take up 136 bits (refer to Chapter IV) and an average message length of 40 bytes was assumed. This would randomly adjust each packet length exponentially.

The next variable parameter is the arrival time of the incoming packets. The most common example of an arrival process is the Poisson process. The average arrival rate of the arrival process - also called the load of the system - is in packets/sec and denoted by the symbol λ. This is simulated in the 'send' state by adding a randomly exponential term (using the op_dist_exponential kernel procedure) to the simulation time as shown:

```
op_intrpt_schedule_self (op_sim_time() + op_dist_exponential(0.1), TX_MSU);
```

Finally, we want to simulate the effects of load sharing in our network of Figure 34. As can be seen in the figure, there are two signaling links to route a message to a particular DPC, i.e., from SSP_1001 (on the left) to SSP_1002 (on the right) or vice versa. The SLS field is used to select a particular signaling link within a given link set, which is a set of links between two SSPs. As stated in Chapter III, the main purpose of load sharing is to keep the load as evenly balanced as possible on the signaling links within a combined link set (which is a load sharing collection of two or more link sets). To simulate the network without load sharing, the following definition is defined in the 'send' state of the ISUP level:

```
op_pk_fd_set(pkptr, FD_INDEX_5TH, OPC_FIELD_TYPE_INTEGER, 5,
              FD_LEN_SLS);
```

where 5 is an arbitrarily assigned SLS number chosen for the simulation. On the other hand, an if-else loop is added to the source code to change the SLS field (from 0 to 15) to simulate the effect of load sharing in the network of Figure 34. This is as shown:

```
if (sls >= 15) {
  sls = 0;
}
else {
  sls = sls + 1;
}
op_pk_fd_set(pkptr, FD_INDEX_5TH, OPC_FIELD_TYPE_INTEGER, sls,
              FD_LEN_SLS);
```

B. SIMULATION RESULTS AND ANALYSIS

In order to study into the total average (steady-state) delay of our network in Figure 34, an OPNET statistic kernel procedure, op_stat_write, is used in the 'get' state of the ISUP level. A state variable, total_handle_delay, is defined and the following executives are defined in the 'get' state:

```
op_stat_write(total_delay_handle, total_delay);
```

Various simulation runs were carried out at different load values for the case of load sharing as well as without load sharing. The results obtained were plotted to obtain the delay-load curve as shown in Figure 35 below.

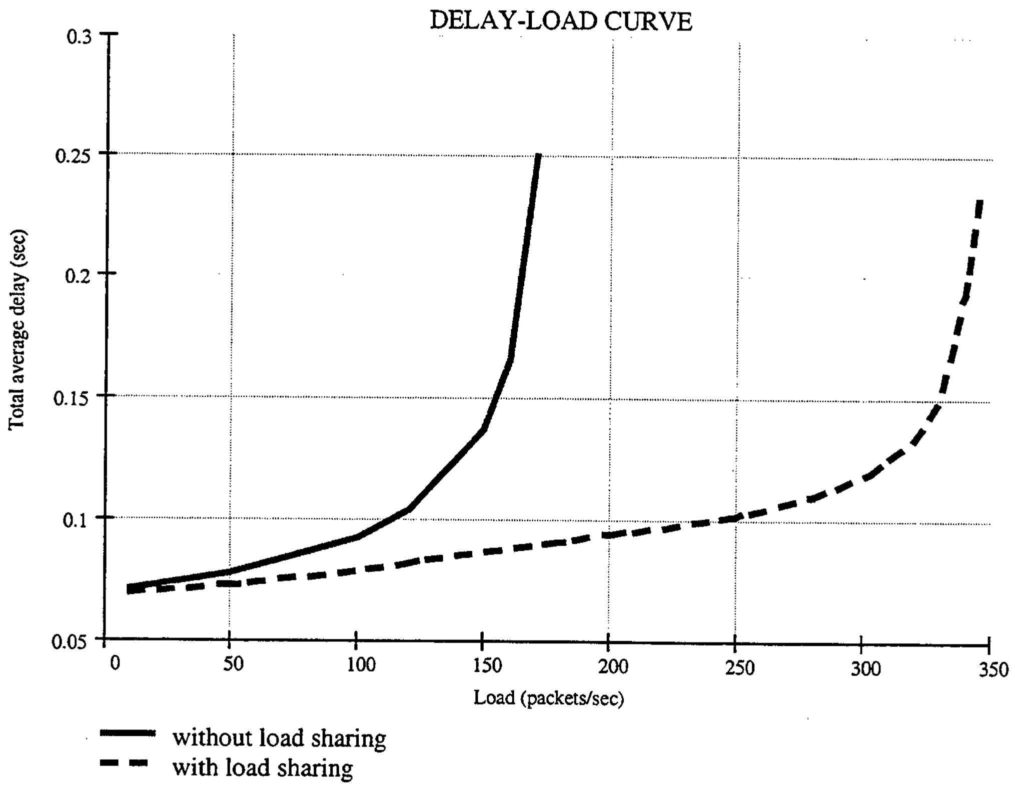


Figure 35. Delay-load curve

For the case of no load sharing, i.e., a single link is used all the time to transmit packages from one SSP to another, the delay increases very significantly as the load approaches 175 packets/sec. This number is expected as the link data rate is 56 kbps and the average packet size is 40 bytes long. On the other hand, for the case of load sharing, since there are now two links a packet can take to reach its destination, the load limit is doubled to 350 packets/sec as can be seen in the dotted curve of Figure 35. Thus, load sharing not only keeps the load as evenly balanced as possible among the signaling links but also increases the load traffic the network can handle. The performance curves obtained in the figure are a typical set of curves encountered in any network that incorporates buffers to store the traffic, which in this case are packets. Hence it can be concluded that the results obtained in our simulation are accurate.

Since the buffers used in the SS7 simulation are infinitely long, the throughput of the network is simply the load of the network. Thus Figure 35 is also the delay-throughput curve. It can seen from the figure that for both curves, the delay remains relatively constant, not increasing very much while the throughput increases significantly, until a certain point where the delay starts to increase significantly while the load or throughput levels off. This point is referred to as the "knee" of the curve and it is also the point at which the ratio of throughput to delay is at a maximum. This ratio is also commonly referred to as the "power" [12]. As our simulated network can be considered an infinite M/M/1 queue network, the average queuing delay is then

$$\text{average\_queuing\_delay} := \frac{1}{\mu}\cdot\frac{1}{1-\rho}$$

where μ represents the average number of packets served per second, or the service rate, and ρ is simply λ/μ. The power is then given by [12] as

$$\text{power} := \frac{\lambda}{\text{average\_queuing\_delay}}$$

$$\text{power} := \rho\cdot\mu^2\cdot(1-\rho)$$

Graphically, the point where a straight line drawn from the origin is tangential to the delay-throughput curve is the knee of the curve or the maximum power. This is indicated in Figure 36 for the case of the curve with load sharing. The same method can also be applied to locate the knee of the curve for the case of no load sharing.

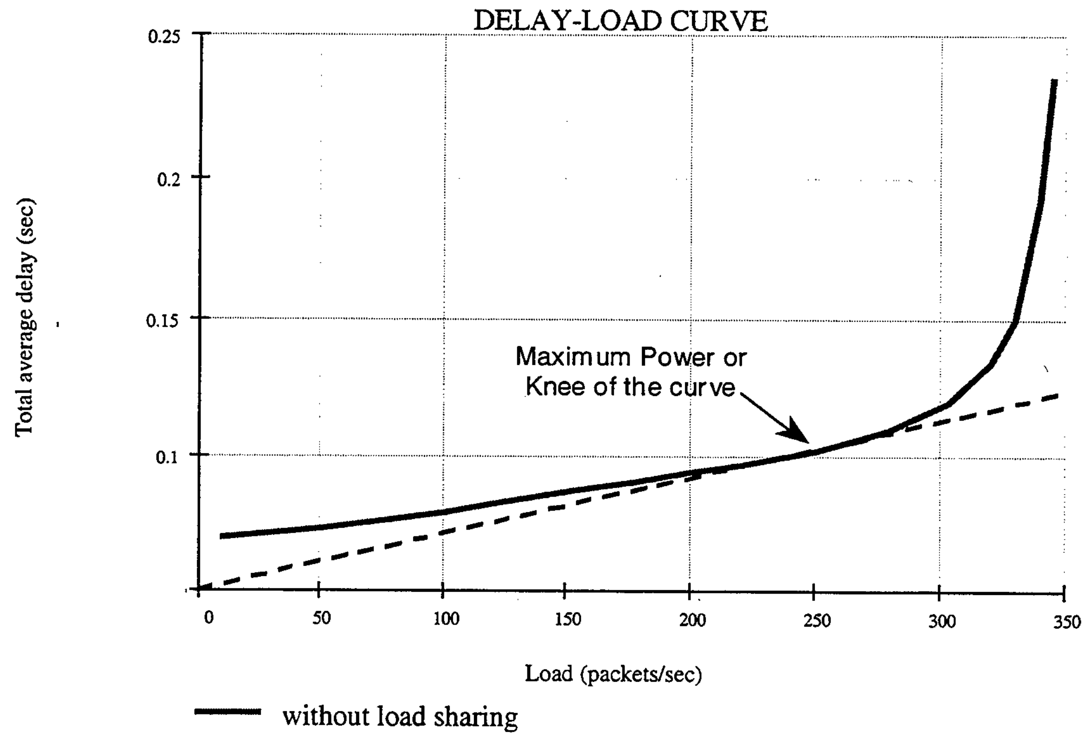


Figure 36. Maximizing power = throughput (or load)/delay

VII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The objective of this thesis was to develop a standard SS7 model in OPNET Modeler that could be built upon to assist in the analysis of related emerging technologies. To meet this objective, a model was implemented that closely follows the algorithms as defined in the International Telecommunication Union (ITU-T) Recommendations for SS7. This thesis focused primarily on the simulation of the SS7's MTP level 3 signaling message handling function.

As defined in SS7, the MTP level 3 comprises of four main modules: signaling message handling, signaling link management, signaling route management, and signaling traffic management. The last three modules perform the operations of the signaling network management function and are not wholly implemented in this study. The signaling message handling module is made up of message discrimination, message routing, and message distribution sub-functions and all these sub-functions are implemented as states in OPNET for the simulation model using the algorithms as defined in ITU-T Recommendation Q.704. Two other states were included in this module, namely, the receive state and the ext_msg state, which are created to handle SS7 messages and all other messages other than SS7 signaling units, respectively.

In order to present a scenario making use of the designed SS7 model, an ISUP level was developed to simulate a simple SS7 network where one SSP is sending ISDN messages (encapsulated on SS7's MSUs) to another SSP, and the total (steady-state) delay was measured and plotted with different traffic loads. Load sharing, a common feature of SS7 network links, was also simulated. The delay-throughput curves obtained closely match those that are encountered in typical networks that incorporate buffers to store traffic.

B. RECOMMENDATIONS FOR FUTURE WORKS

The modeling of the standard SS7 kernel in this thesis was designed to be a starting point, providing a foundation upon which future studies looking into SS7's impact on operational Department of Defense (DoD) telecommunications projects and other related and evolving technologies can be carried out with little difficulty. One such study could be the transporting of SS7 messages between PSTN and Internet Protocol (IP) based networks, such as IP Telephony. This is the direction that future communications networks appears to be heading as the convergence of the traditional PSTN circuit-switched technology and IP data-packet technology creates capabilities that neither network can support on its own. To do this study, first, an IP network consisting of IP servers and routers must be simulated in OPNET. The simulation of the IP server will require the development of various processes, such as an Ethernet port driver (using OPNET transmitter and receiver), Address Resolution Protocol (ARP), IP incorporating the encapsulation and decapsulation functions, User Datagram Protocol (UDP) or Transmission Control Protocol (TCP) and user process. The development of the IP router will be almost similar to the server except there are more Ethernet ports linked to the IP process. Standardized protocols used for call control within the IP network and between IP-based networks and PSTN are still being studied by a number of organizations. A single standard is unlikely to be agreed upon for quite some time. Consequently, the second step may involve the need to develop (in OPNET) a simple IP Telephony switch that comprises H.323 gatekeeper and H.323 gateway. H.323 is the ITU-T Recommendation for visual telephone systems and equipment for Local Area Networks that provide a non-guaranteed Quality of Service. Interested readers are encouraged to refer to [13] for more details on the implementation.

It can be concluded from the simulation work done in this thesis that the OPNET software suite is an excellent modeling tool for developing a standard SS7 kernel and using it to study its impact on existing telecommunications projects as well as analyze emerging technologies.

APPENDIX A. MTP LEVEL 3 HEADER BLOCK

In this and the following appendices, the source codes for implementing SS7's MTP level 3 are listed. Comments are provided where appropriate in the codes.

HEADER BLOCK

```
/* ==================================================================== */
/* Signaling System 7 (SS7), Message Transfer Part Level 3 (MTP3)       */
/* ITU-T Recommendation Q.704, Jul 1996                                 */
/* The definitions here are for these purposes                          */
/*    (1) General constants                                             */
/*    (2) I/O Stream used for communication with External Modules       */
/*    (3) Queuing buffers allocation and use                            */
/*    (4) Internal Message Codes                                        */
/*    (5) Link and States of the Level 2 Functional Blocks              */
/*    (6) Interrupt Codes and Interrupt Macros                          */
/*    (7) Timer Data Type                                               */
/* ==================================================================== */

/* ==================================================================== */
/* (1) General constants                                                */
/* ==================================================================== */

/* define TRUE and FALSE */
#define TRUE  1
#define FALSE 0

/* define modular 128 */
#define MOD128 128

/* define signaling point types */
#define SP      0
#define STP     1

/* define system limits */
#define SYS_LINK_LIMIT    16    /* define max number of links to one SP */
#define SYS_ADJ_SP_LIMIT  16    /* define max number of adjacent SP     */
#define SYS_ROUTE_LIMIT   1000  /* define max number of routes assumed  */
#define SYS_MAX_HOPS      3

/* define MTP3 configuration parameters */
#define SLM_SIGNAL_TERMINAL_PRE_DETERMINED TRUE
#define SLM_DATA_LINK_PRE_DETERMINED       TRUE

/* ==================================================================== */
/* (2) I/O Stream used for communication with External Modules         */
/*     Note: Input/Output streams for communication with MTP2 are       */
/*           allocated from 11 to 42. The gdf files defining the        */
/*           network configuration would have to used this range        */
/* ==================================================================== */

/* define MTP3 Input Streams */
#define SCCP_INPUT_STREAM           0
```

```
#define TEL_UP_INPUT_STREAM           1
#define ISDN_UP_INPUT_STREAM          2
#define DATA_UP_CCR_INPUT_STREAM      3
#define DATA_UP_FRC_INPUT_STREAM      4
#define BISDN_UP_INPUT_STREAM         5
#define SAT_ISDN_UP_INPUT_STREAM      6
#define MTP_TESTING_UP_INPUT_STREAM   7
#define NET_TEST_MAINT_INPUT_STREAM   8
#define MTP3_RETRIEVE_INPUT_STREAM    99

/* define MTP3 output Streams */
#define SCCP_OUTPUT_STREAM            0
#define TEL_UP_OUTPUT_STREAM          1
#define ISDN_UP_OUTPUT_STREAM         2
#define DATA_UP_CCR_OUTPUT_STREAM     3
#define DATA_UP_FRC_OUTPUT_STREAM     4
#define BISDN_UP_OUTPUT_STREAM        5
#define SAT_ISDN_UP_OUTPUT_STREAM     6
#define MTP_TESTING_UP_OUTPUT_STREAM 7
#define NET_TEST_MAINT_OUTPUT_STREAM 8

/* ================================================================ */
/* (3) Queuing buffers allocation and use                           */
/*     Note: The buffer numbers used are the OPNET "Queue" module's sub */
/*           -queue number. The Transfer buffer is for queuing out-     */
/*           going signaling units, while the Retransmission buffer     */
/*           stores every sent unit till an ackowledegment of receipt   */
/*           has been recieved. Other buffers are for queuing internal  */
/*           messages for each of the level 2 functional blocks.        */
/* ================================================================ */

#define HMDC_MSU_BUFFER 0
#define HMDT_MSU_BUFFER 1
#define HMRT_MSU_BUFFER 2
#define HMDC_BUFFER 3
#define HMDT_BUFFER 4
#define HMRT_BUFFER 5
#define TLAC_BUFFER 6
#define TSRC_BUFFER 7
#define TCOC_BUFFER 8
#define TCBC_BUFFER 9
#define TFRC_BUFFER 10
#define TCRC_BUFFER 11
#define TSFC_BUFFER 12
#define TPRC_BUFFER 13
#define LLSC_BUFFER 14
#define LSAC_BUFFER 15
#define LSLA_BUFFER 16
#define LSLR_BUFFER 17
#define LSLD_BUFFER 18
#define LSTA_BUFFER 19
#define LSDA_BUFFER 20
#define RTPC_BUFFER 21
#define RTAC_BUFFER 22
#define RSRT_BUFFER 23
#define RTCC_BUFFER 24
```

```
#define RCAT_BUFFER 25
#define RTRC_BUFFER 26


/* ================================================================== */
/* (4) Field Constants                                                */
/*     Note: Values used are as defined in ITU-T Q.704 Jul 96         */
/* ================================================================== */

/* define field index for use in OPNET */
#define FD_INDEX_1ST   1
#define FD_INDEX_2ND   2
#define FD_INDEX_3RD   3
#define FD_INDEX_4TH   4
#define FD_INDEX_5TH   5
#define FD_INDEX_6TH   6
#define FD_INDEX_7TH   7
#define FD_INDEX_8TH   8
#define FD_INDEX_9TH   9
#define FD_INDEX_10TH  10

/* define size of SS7 packet fields */
#define FD_LEN_FLAG     8  /* size of header and trailer Flag   */
#define FD_LEN_BSN      7  /* size of Backward Sequence Number  */
#define FD_LEN_BIB      1  /* size of Backward Indicator Bit    */
#define FD_LEN_FSN      7  /* size of Forward Sequence Number   */
#define FD_LEN_FIB      1  /* size of Forward Indicator Bit     */
#define FD_LEN_LI       8  /* size of Length Indicator (6 used) */
#define FD_LEN_CK       16 /* size of check bits                */
#define FD_LEN_PDU      -1 /* size of PDU set to -1 for encap   */
#define FD_LEN_SF       8  /* size of Status Field              */
#define FD_LEN_SIO      8  /* size of Service Information Octet */
#define FD_LEN_SI       4  /* size of Service Indicator         */
#define FD_LEN_SSF      4  /* size of Sub-Service Field         */
#define FD_LEN_DPC      14 /* size of Destination Point Code    */
#define FD_LEN_OPC      14 /* size of originating Point Code    */
#define FD_LEN_SLS      4  /* size of signaling link selection  */
#define FD_LEN_H0H1     8  /* size of H0H1 field                */
#define FD_LEN_SPARE2   2  /* size of spare field with 2 bits   */
#define FD_LEN_LABEL    32 /* size of a standard label          */
#define FD_LEN_PC       14 /* size of a point code              */

#define FLAG 126

/* definition of SS7 MTP Level 3 Service Indicators (SI) */
/* (international signaling network)                     */
#define SIO_SI_SIG_NET_MGMT    0x0
#define SIO_SI_NET_TEST_MAINT  0x1
#define SIO_SI_SCCP            0x3
#define SIO_SI_TEL_UP          0x4
#define SIO_SI_ISDN_UP         0x5
#define SIO_SI_DATA_UP_CCR     0x6
#define SIO_SI_DATA_UP_FRC     0x7
#define SIO_SI_MTP_TESTING_UP  0x8
#define SIO_SI_BISDN_UP        0x9
#define SIO_SI_SAT_ISDN_UP     0xA
```

```
/* definition of SS7 MTP Level 3 Sub-service Field (SSF)  */
#define SIO_SSF_INTERNATIONAL  0x0
#define SIO_SSF_NATIONAL       0x8

/* definition H0 and H1 codes for Network Management MSUs */
#define MSG_SNM_CHANGEOVER_ORDER                         0x11
#define MSG_SNM_CHANGEOVER_ACK                           0x21
#define MSG_SNM_CHANGEBACK_DECLARATION                   0x51
#define MSG_SNM_CHANGEBACK_ACK                           0x61
#define MSG_SNM_EMERGENCY_CHANGEOVER_ORDER               0x12
#define MSG_SNM_EMERGENCY_CHANGEOVER_ACK                 0x22
#define MSG_SNM_TRANSFER_PROHIBITED                      0x14
#define MSG_SNM_TRANSFER_ALLOWED                         0x54
#define MSG_SNM_TRANSFER_RESTRICTED                      0x34
#define MSG_SNM_SIG_ROUTE_SET_TEST_PROHIBITED            0x15
#define MSG_SNM_SIG_ROUTE_SET_TEST_RESTRICTED            0x25
#define MSG_SNM_LINK_INHIBIT                             0x16
#define MSG_SNM_LINK_UNINHIBIT                           0x26
#define MSG_SNM_LINK_INHIBIT_ACK                         0x36
#define MSG_SNM_LINK_UNINHIBIT_ACK                       0x46
#define MSG_SNM_LINK_INHIBIT_DENIED                      0x56
#define MSG_SNM_LINK_FORCE_UNINHIBIT                     0x66
#define MSG_SNM_LINK_LOCAL_INHIBIT_TEST                  0x76
#define MSG_SNM_LINK_REMOTE_INHIBIT_TEST                 0x86
#define MSG_SNM_TRAFFIC_RESTART_ALLOWED                  0x17
#define MSG_SNM_DATALINK_CONNECTION_ORDER                0x18
#define MSG_SNM_DATALINK_CONNECTION_SUCCESSFUL           0x28
#define MSG_SNM_DATALINK_CONNECTION_NOT_SUCCESSFUL       0x38
#define MSG_SNM_DATALINK_CONNECTION_NOT_POSSIBLE         0x48
#define MSG_SNM_TRANSFER_CONTROLLED                      0x23
#define MSG_SNM_SIG_ROUTE_SET_CONGESTION_TEST            0x13
#define MSG_SNM_USER_PART_UNAVAILABLE                    0x1A
#define CODE_H0H1_SNTM_SIG_LINK_TEST_CONTROL_MSG         0x00

/*Definition of the User Part Identity for the User Part Unavailable Message*/
#define MSG_USER_PART_SCCP                    0x3
#define MSG_USER_PART_TUP                     0x4
#define MSG_USER_PART_ISUP                    0x5
#define MSG_USER_PART_DUP                     0x6
#define MSG_USER_PART_MTP_TEST_UP             0x8
#define MSG_USER_PART_BISDN                   0x9
#define MSG_USER_PART_SISDN                   0xA

/* Definition of the Unavailability Cause for the User Part Unavailable
   Message */
#define CODE_UP_UNKNOWN                       0x0
#define CODE_UP_UNEQUIPPED_RU                 0x1
#define CODE_UP_INACCESSIBLE_RU               0x2
```

```
/* ====================================================================== */
/* (5) Internal MTP3 Messages                                             */
/* ====================================================================== */

#define MSG_TPRC_HMDC_RESTART_BEGINS                            1001
#define MSG_TPRC_HMDC_RESTART_ENDS                              1002

#define MSG_TPRC_HMDT_RESTART_BEGINS                            1201
#define MSG_TPRC_HMDT_RESTART_ENDS                              1202
#define MSG_TSRC_HMDT_ADJ_SP_RESTART                            1203

#define MSG_TPRC_HMRT_RESTART_BEGINS                            1301
#define MSG_TPRC_HMRT_RESTART_ENDS                              1302
#define MSG_TLAC_HMRT_INHIBIT_DENIED                            1303
#define MSG_TLAC_HMRT_LINK_INHIBIT                              1304
#define MSG_TLAC_HMRT_INHIBIT_ACK                               1305
#define MSG_TLAC_HMRT_UNINHIBIT_LINK                            1306
#define MSG_TLAC_HMRT_EMERGENCY_CHANGEOVER_ACK                  1307
#define MSG_TLAC_HMRT_UNINHIBIT_ACK                             1308
#define MSG_TLAC_HMRT_FORCE_UNINHIBIT_LINK                      1309
#define MSG_TCRC_HMRT_UPDATE_ROUTING_TABLES                     1310
#define MSG_TFRC_HMRT_UPDATE_ROUTING_TABLES                     1311

#define MSG_HMDT_TLAC_LINK_INHIBIT                              1401
#define MSG_HMDT_TLAC_LINK_INHIBIT_DENIED                       1402
#define MSG_HMDT_TLAC_LINK_INHIBIT_ACK                          1403
#define MSG_HMDT_TLAC_CHANGEOVER_ORDER                          1404
#define MSG_HMDT_TLAC_EMERGENCY_CHANGEOVER_ORDER                1405
#define MSG_HMDT_TLAC_LINK_FORCE_UNINHIBIT                      1406
#define MSG_HMDT_TLAC_LINK_UNINHIBIT                            1407
#define MSG_HMDT_TLAC_LINK_UNINHIBIT_ACK                        1408
#define MSG_HMDT_TLAC_LINK_LOCAL_INHIBIT_TEST                   1409
#define MSG_HMDT_TLAC_LINK_REMOTE_INHIBIT_TEST                  1410
#define MSG_LSAC_TLAC_REMOTE_PRO_OUTAGE                         1501
#define MSG_LSAC_TLAC_REMOTE_PRO_RECOVERED                      1502
#define MSG_LSAC_TLAC_LINK_IN_SERVICE                           1503
#define MSG_LSAC_TLAC_LINK_FAILED                               1504

#define MSG_HMDT_TCOC_CHANGEOVER_ACK                            1601
#define MSG_HMDT_TCOC_EMERGENCY_CHANGEOVER_ACK                  1602

#define MSG_HMDT_TCBC_CHANGEBACK_DECLARATION                    1701
#define MSG_HMDT_TCBC_CHANGEBACK_ACK                            1702

#define MSG_HMDT_TPRC_TRAFFIC_RESTART_ALLOWED                   1801

#define MSG_HMRT_TSFC_MSG_FOR_CONGESTED_DEST                    1901

#define MSG_TPRC_LLSC_RESTART_BEGINS                            4001
#define MSG_TPRC_LLSC_RESTART_ENDS                              4002
#define MSG_LSAC_LLSC_LINK_ACTIVE                               4003
#define MSG_LSAC_LLSC_LINK_INACTIVE                             4004
#define MSG_LSAC_LLSC_LINK_FAILED                               4005
#define MSG_LSAC_LLSC_ACTIVATE_ANOTHER_LINK                     4006
#define MSG_TSRC_LLSC_EMERGENCY                                 4007
```

```
#define MSG_TSRC_LLSC_EMERGENCY_CEASES                          4008
#define MSG_MGMT_LLSC_ACTIVATE_LINK_SET                         4009
#define MSG_MGMT_LLSC_DEACTIVATE_LINK_SET                       4010

#define MSG_TLAC_LSAC_CONTINUE                                  4101
#define MSG_TLAC_LSAC_FLUSH_BUFFERS                             4102
#define MSG_TLAC_LSAC_CHANGEOVER_ORDER_RECEIVED                 4103
#define MSG_TCOC_LSAC_STM_READY                                 4104
#define MSG_TCOC_LSAC_STOP_L2                                   4105
#define MSG_LLSC_LSAC_ACTIVATE_LINK                             4106
#define MSG_LLSC_LSAC_DEACTIVATE_LINK                           4107
#define MSG_LLSC_LSAC_EMERGENCY                                 4108
#define MSG_LLSC_LSAC_EMERGENCY_CEASES                          4109
#define MSG_LLSC_LSAC_RESTART_ENDS                              4110
#define MSG_LSLA_LSAC_START_SIG_LINK                            4111
#define MSG_LSLA_LSAC_ACTIVATION_UNSUCCESSFUL                   4112
#define MSG_LSLR_LSAC_START_SIG_LINK                            4113
#define MSG_LSLR_LSAC_RESTORATION_UNSUCCESSFUL                  4114
#define MSG_SLTC_LSAC_SLT_SUCCESSFUL                            4115
#define MSG_SLTC_LSAC_SLT_FAILED                                4116
#define MSG_MGMT_LSAC_ACTIVATE_LINK                             4117
#define MSG_MGMT_LSAC_DEACTIVATE_LINK                           4118

#define MSG_LSAC_LSLA_START_ACTIVATION                          4201
#define MSG_LSAC_LSLA_RESTART_ACTIVATION                        4202
#define MSG_LSAC_LSLA_DEACTIVATE_LINK                           4203

#define MSG_LSAC_LSLR_START_RESTORATION                         4301
#define MSG_LSAC_LSLR_RESTART_RESTORATION                       4302

#define MSG_LSAC_LSLD_DEACTIVATE_LINK                           4401

#define MSG_HMDT_LSDA_DATALINK_CONNECTION_ORDER                 4501
#define MSG_HMDT_LSDA_DATALINK_CONNECTION_SUCCESSFUL            4502
#define MSG_HMDT_LSDA_DATALINK_CONNECTION_NOT_SUCCESSFUL        4503
#define MSG_HMDT_LSDA_DATALINK_CONNECTION_NOT_POSSIBLE          4504

#define MSG_HMDT_RSRT_SIG_ROUTE_SET_TEST_PROHIBITED             5101
#define MSG_HMDT_RSRT_SIG_ROUTE_SET_TEST_RESTRICTED             5102

#define MSG_HMRT_RTPC_MSG_RECEIVED_FOR_INACCESSIBLE_SP          5201
#define MSG_HMDT_RTPC_TRANSFER_PROHIBITED                       5202

#define MSG_HMDT_RTAC_TRANSFER_ALLOWED                          5301

#define MSG_HMDT_RTRC_TRANSFER_RESTRICTED                       5401

#define MSG_HMDT_RTCC_TRANSFER_CONTROLLED                       5501
#define MSG_HMDT_RTCC_USER_PART_AVAILABLE                       5502

#define MSG_HMDT_RCAT_SIG_ROUTE_SET_CONGESTION_TEST             5601

#define MSG_LSAC_LSC_EMERGENCY                                  8001
#define MSG_LSAC_LSC_EMERGENCY_CEASES                           8002
#define MSG_LSAC_LSC_STOP                                       8003
#define MSG_LSAC_LSC_START                                      8004
```

```
#define MSG_LSAC_LSC_FLUSH_BUFFERS                                8005
#define MSG_LSAC_LSC_CONTINUE                                     8006
#define MSG_LSAC_LSC_LOCAL_PRO_OUTAGE                             8007
#define MSG_LSAC_LSC_LOCAL_PRO_RECOVERED                          8008


#define MSG_LSC_LSAC_IN_SERVICE                                   8161
#define MSG_LSC_LSAC_OUT_SERVICE                                  8162
#define MSG_LSC_LSAC_REMOTE_PRO_OUTAGE                            8163
#define MSG_LSC_LSAC_REMOTE_PRO_RECOVERED                         8164

/* ===================================================================== */
/* State/Status Definitions                                               */
/* ===================================================================== */

/* define the link status */
/* states and values assigned are as per SS7 MTP2, Q.703 */
#define LINK_STATUS_OUT_ALIGNMENT       0
#define LINK_STATUS_NORMAL              1
#define LINK_STATUS_EMERGENCY           2
#define LINK_STATUS_OUT_SERVICE         3
#define LINK_STATUS_PROCESSOR_OUTAGE    4
#define LINK_STATUS_BUSY                5

#define LINKSET_INACTIVE  0
#define LINKSET_ACTIVE    1

#define LINK_INACTIVE                   0
#define LINK_ACTIVATING_RESTORING       1
#define LINK_ACTIVATING_RESTORING_WAIT  2
#define LINK_ACTIVE                     3
#define LINK_ACTIVE_WAIT                4

/* define internal states for SMH */
/* states assigned are as per SS7, MTP3, Q.704 */
#define STATE_HMDC_IDLE                  0
#define STATE_HMDC_OWN_SP_RESTARTING     1
#define STATE_HMDT_IDLE                  0
#define STATE_HMDT_OWN_SP_RESTARTING     1
#define STATE_HMRT_IDLE                  0
#define STATE_HMRT_OWN_SP_RESTARTING     1

/* define internal states for STM */
/* states assigned are as per SS7, MTP3, Q.704 */
#define STATE_TLAC_UNAVAILABLE         0
#define STATE_TLAC_AVAILABLE           1
#define STATE_TLAC_SP_RESTARTING       2

/* define internal states for SLM */
/* states assigned are as per SS7, MTP3, Q.704 */
#define STATE_LSLA_IDLE                0
#define STATE_LSLR_IDLE                0
#define STATE_LSLD_IDLE                0
```

```
/* define internal states for SRM */
/* states assigned are as per SS7, MTP3, Q.704 */
#define STATE_RTPC_IDLE              0
#define STATE_RTPC_WAIT              1
#define STATE_RTAC_IDLE              0

/* ================================================================ */
/* Interrupt Definitions                                            */
/* ================================================================ */

/* definition of self interrupt codes */
#define HMDC_MSG          1
#define HMDT_MSG          2
#define HMRT_MSG          3
#define TLAC_MSG          4
#define TSRC_MSG          5
#define TPRC_MSG          6
#define TSFC_MSG          7
#define TCOC_MSG          8
#define TCBC_MSG          9
#define TFRC_MSG          10
#define TCRC_MSG          11
#define LLSC_MSG          12
#define LSAC_MSG          13
#define LSLA_MSG          14
#define LSLR_MSG          15
#define LSLD_MSG          16
#define LSDA_MSG          17
#define LSTA_MSG          18
#define RTPC_MSG          19
#define RTAC_MSG          20
#define RTRC_MSG          21
#define RSRT_MSG          22
#define RTCC_MSG          23
#define RCAT_MSG          24
#define HMDC_MSU          51
#define HMDT_MSU          52
#define HMRT_MSU          53
#define T17_EXPIRE        117

/* definition of interrupts processed */
#define HMDC  ((op_intrpt_type() == OPC_INTRPT_SELF) && ((op_intrpt_code() ==
HMDC_MSG) || (op_intrpt_code() == HMDC_MSU)))
#define HMDT  ((op_intrpt_type() == OPC_INTRPT_SELF) && ((op_intrpt_code() ==
HMDT_MSG) || (op_intrpt_code() == HMDT_MSU)))
#define HMRT  ((op_intrpt_type() == OPC_INTRPT_SELF) && ((op_intrpt_code() ==
HMRT_MSG) || (op_intrpt_code() == HMRT_MSU)))
#define TLAC  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
TLAC_MSG))
#define TSRC  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
TSRC_MSG))
#define TPRC  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
TPRC_MSG))
#define TSFC  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
TSFC_MSG))
```

```
#define TCOC  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
TCOC_MSG))
#define TCBC  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
TCBC_MSG))
#define TFRC  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
TFRC_MSG))
#define TCRC  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
TCRC_MSG))
#define LLSC  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
LLSC_MSG))
#define LSAC  ((op_intrpt_type() == OPC_INTRPT_SELF) && ((op_intrpt_code() ==
LSAC_MSG) || (op_intrpt_code() == T17_EXPIRE)))
#define LSLA  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
LSLA_MSG))
#define LSLR  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
LSLR_MSG))
#define LSLD  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
LSLD_MSG))
#define LSDA  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
LSDA_MSG))
#define LSTA  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
LSTA_MSG))
#define RTPC  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
RTPC_MSG))
#define RTAC  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
RTAC_MSG)).
#define RTRC  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
RTRC_MSG))
#define RSRT  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
RSRT_MSG))
#define RTCC  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
RTCC_MSG))
#define RCAT  ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code() ==
RCAT_MSG))
#define MSU_ARRIVAL  (op_intrpt_type() == OPC_INTRPT_STRM)
#define EXTERNAL_MSG (op_intrpt_type() == OPC_INTRPT_REMOTE)


/* Definition of a standard timer data structure */
typedef struct {
   Evhandle ev;      /* The OPNET event handle for the self interrupt */
   int      code;    /* A code value to be returned by the interrupt  */
   double   delay;   /* The time before the timer expires             */
   char     active;  /* True if the timer is running                  */
   int      data;    /* extra data field related to the timer         */
} timer_type;

/* define user availablity data structure */
typedef struct {
  int available;
  int user_part_id;
  int unavail_cause;
} user_avail_type;
```

```
typedef struct {
   int subnet_user_id;
   int user_id;
   int point_code;
   int type;
} node_type;

typedef struct {
   int connected;
   int link_slc;
   int stream_num;
   int SP_connected;
} master_link_type;

/* routing tables for HMRT */
typedef struct {
   int connected;                  /* link is connected in Opnet      */
   int link_slc;                   /* link number                     */
   int stream_num;                 /* I/O stream number for this link */
   int SP_connected;               /* adjSP connected                 */
   int adjSP_restarting;           /* adjSP restarting                */
   int link_available;             /* link is available               */
} HMRT_link_type;

typedef struct {
   int subnet_user_id;             /* subnet user id of dest node     */
   int user_id;                    /* user id of dest node            */
   int point_code;                 /* point code of the dest node     */
   int node_type;                  /* type of destination, SP or STP  */
   int dest_accessible;            /* destination accessible          */
   int dest_congested;             /* destination congested           */
   List * route_list_ptr;          /* pointer to all possible routes  */
   int route_num;                  /* number of all possible routes   */
} HMRT_dest_type;

typedef struct {
  int nextSP;                      /* point code of next SP           */
  int route_available;             /* route is available              */
  List * sls_assign_list_ptr;      /* list of sls assigned            */
} HMRT_route_type;

typedef struct {
  int sls_num;                     /* sig selection number            */
  int link_slc;                    /* link for this number            */
} HMRT_sls_assign_type;

/* routing tables format for Signaling Link Management */
typedef struct {
   int connected;                  /* link is connected for use       */
   int link_slc;                   /* link number                     */
   int stream_num;                 /* I/O stream number for this link */
   int SP_connected;               /* adjSP connected                 */
   Objid link_objid;               /* Opnet object id of adjacent SP  */
   int linkset_status;             /* linkset active status           */
   int emergency;                  /* emergency flag                  */
   int active_status;              /* link activattion status         */
   int activation;               /* activation flag                  */
   int first_failure;            /* first failure flag               */
   int act_rest_unsuccessful;    /* activation unsuccessful flag     */
} SLM_link_type;


/* funtcion prototypes */

int mtp3_increment(int value, int step, int mod_number);
int mtp3_decrement(int value, int step, int mod_number);
void ss7_mtp3_timer_warn(char * msg);
void mtp3_timer_start(timer_type * timer);
void mtp3_timer_stop(timer_type * timer);
Objid MTP3_from_MTP2(Objid objid);
void mtp3_send_external_msg(Objid target_id, int msg_code, int msg_data,
                            Ici * iciptr);
void discard_MSU(Packet * pkptr);

HMRT_sls_assign_type * sf_rt_HMRT_pick_a_link_for_tx(int destSP, int sls);
HMRT_sls_assign_type * sf_rt_HMRT_pick_a_link_in_linkset_for_tx(int destSP);
HMRT_sls_assign_type * sf_rt_HMRT_pick_any_link_for_tx();
HMRT_sls_assign_type * sf_rt_HMRT_pick_a_direct_link_for_tx(int destSP, int
sls);
void sf_rt_HMRT_update_routing_tables();
int sf_rt_HMRT_is_route_available(int destSP);
int sf_rt_HMRT_is_DPC_adj_and_restarting(int destSP);
int sf_rt_HMRT_is_DPC_congested(int destSP);
int sf_rt_SLM_linkset_status(int adjSP);
void sf_rt_SLM_set_linkset_active(int adjSP);
void sf_rt_SLM_cancel_linkset_active(int adjSP);

void SNM_CHANGEOVER_ORDER(Packet * pkptr);
void SNM_CHANGEOVER_ACK(Packet * pkptr);
void SNM_CHANGEBACK_DECLARATION(Packet * pkptr);
void SNM_CHANGEBACK_ACK(Packet * pkptr);
void SNM_EMERGENCY_CHANGEOVER_ORDER(Packet * pkptr);
void SNM_EMERGENCY_CHANGEOVER_ACK(Packet * pkptr);
void SNM_TRANSFER_PROHIBITED(Packet * pkptr);
void SNM_TRANSFER_ALLOWED(Packet * pkptr);
void SNM_TRANSFER_RESTRICTED(Packet * pkptr);
void SNM_SIG_ROUTE_SET_TEST_PROHIBITED(Packet * pkptr);
void SNM_SIG_ROUTE_SET_TEST_RESTRICTED(Packet * pkptr);
void SNM_LINK_INHIBIT(Packet * pkptr);
void SNM_LINK_UNINHIBIT(Packet * pkptr);
void SNM_LINK_INHIBIT_ACK(Packet * pkptr);
void SNM_LINK_UNINHIBIT_ACK(Packet * pkptr);
void SNM_LINK_INHIBIT_DENIED(Packet * pkptr);
void SNM_LINK_FORCE_UNINHIBIT(Packet * pkptr);
void SNM_LINK_LOCAL_INHIBIT_TEST(Packet * pkptr);
void SNM_LINK_REMOTE_INHIBIT_TEST(Packet * pkptr);
void SNM_TRAFFIC_RESTART_ALLOWED(Packet * pkptr);
void SNM_DATALINK_CONNECTION_ORDER(Packet * pkptr);
void SNM_DATALINK_CONNECTION_SUCCESSFUL(Packet * pkptr);
void SNM_DATALINK_CONNECTION_NOT_SUCCESSFUL(Packet * pkptr);
void SNM_DATALINK_CONNECTION_NOT_POSSIBLE(Packet * pkptr);
void SNM_TRANSFER_CONTROLLED(Packet * pkptr);
void SNM_SIG_ROUTE_SET_CONGESTION_TEST(Packet * pkptr);
void SNM_USER_PART_UNAVAILABLE(Packet * pkptr);

void TPRC_LLSC_RESTART_BEGINS(int adjSP);
void TPRC_LLSC_RESTART_ENDS(int adjSP);
void LSAC_LLSC_LINK_ACTIVE(int adjSP);
void LSAC_LLSC_LINK_INACTIVE(int adjSP);
void LSAC_LLSC_LINK_FAILED(int adjSP);
void LSAC_LLSC_ACTIVATE_ANOTHER_LINK(int adjSP);
void TSRC_LLSC_EMERGENCY(int adjSP);
void TSRC_LLSC_EMERGENCY_CEASES(int adjSP);

void TLAC_LSAC_CONTINUE(int link_num);
void TLAC_LSAC_FLUSH_BUFFERS(int link_num);
void TLAC_LSAC_CHANGEOVER_ORDER_RECEIVED(int link_num);
void TCOC_LSAC_STM_READY(int link_num);
void TCOC_LSAC_STOP_L2(int link_num);
void LLSC_LSAC_ACTIVATE_LINK(int link_num);
void LLSC_LSAC_DEACTIVATE_LINK(int link_num);
void LLSC_LSAC_EMERGENCY(int link_num);
void LLSC_LSAC_EMERGENCY_CEASES(int link_num);
void LLSC_LSAC_RESTART_ENDS(int link_num);
void LSLA_LSAC_START_SIG_LINK(int link_num);
void LSLA_LSAC_ACTIVATION_UNSUCCESSFUL(int link_num);
void LSLR_LSAC_START_SIG_LINK(int link_num);
void LSLR_LSAC_RESTORATION_UNSUCCESSFUL(int link_num);
void SLTC_LSAC_SLT_SUCCESSFUL(int link_num);
void SLTC_LSAC_SLT_FAILED(int link_num);
void LSC_LSAC_IN_SERVICE(int link_num);
void LSC_LSAC_OUT_SERVICE(int link_num);
void LSC_LSAC_REMOTE_PRO_RECOVERED(int link_num);
void LSC_LSAC_REMOTE_PRO_OUTAGE(int link_num);

void LSAC_LSLA_START_ACTIVATION(int link_num);
void LSAC_LSLA_RESTART_ACTIVATION(int link_num);
void LSAC_LSLA_DEACTIVATE_LINK(int link_num);

void LSAC_LSLR_START_RESTORATION(int link_num);
void LSAC_LSLR_RESTART_RESTORATION(int link_num);

void LSAC_LSLD_DEACTIVATE_LINK(int link_num);

void LSAC_TLAC_REMOTE_PRO_OUTAGE(int link_num);
void LSAC_TLAC_REMOTE_PRO_RECOVERED(int link_num);
void LSAC_TLAC_LINK_IN_SERVICE(int link_num);
void LSAC_TLAC_LINK_FAILED(int link_num);

void HMRT_TSFC_MSG_FOR_CONGESTED_DEST(int orig_sp, int destSP);

void HMRT_RTPC_MSG_RECEIVED_FOR_INACCESSIBLE_SP(int origSP, int destSP);
void HMRT_MGMT_MSG_RECEIVED_FOR_INACCESSIBLE_SP(int origSP, int destSP);
void LSAC_SLTC_START_SLT(int link_num);
```

APPENDIX B. MTP LEVEL 3 STATE VARIABLES

STATE VARIABLES BLOCK

```
/* Own Object Id */
Objid \own_objid;

/* Own Subnet's User id */
int   \own_subnet_user_id;

/* Own User id */
int   \own_user_id;

/* Own Destination Point Code */
int   \own_PC;

/* Own SP type, SP or STP */
int   \own_SP_type;

/* pointer to Opnet's network topology */
Topology *  \network_ptr;

/* List pointer to all the nodes */
List *      \node_table_ptr;

/* Total number of nodes in network */
int   \node_total;

/* master table on link connected */
master_link_type  \master_link_table[SYS_LINK_LIMIT];

/* total number of links connected */
int   \link_total;

/* user part availability table */
user_avail_type   \user_availability_table[16];

/* definie ICI variables */
Ici * \iciptr_external;

/* service time for HMDC MSU */
double      \HMDC_MSU_service_time;

/* service time for HMDT MSU */
double      \HMDT_MSU_service_time;

/* service time for HMRT MSU */
double      \HMRT_MSU_service_time;

/* service time for HMDC messages */
double      \HMDC_service_time;
```

```
/* service time for HMDT messages */
double        \HMDT_service_time;

/* service time for HMRT messages */
double        \HMRT_service_time;

/* service time for TLAC messages */
double        \TLAC_service_time;

/* service time for TSRC messages */
double        \TSRC_service_time;

/* service time for TPRC messages */
double        \TPRC_service_time;

/* service time for TSFC messages */
double        \TSFC_service_time;

/* service time for TCOC messages */
double        \TCOC_service_time;

/* service time for TCBC messages */
double        \TCBC_service_time;

/* service time for TFRC messages */
double        \TFRC_service_time;

/* service time for TCRC messages */
double        \TCRC_service_time;

/* service time for LLSC messages */
double        \LLSC_service_time;

/* service time for LSAC messages */
double        \LSAC_service_time;

/* service time for LSLA messages */
double        \LSLA_service_time;

/* service time for LSLR messages */
double        \LSLR_service_time;

/* service time for LSLD messages */
double        \LSLD_service_time;

/* service time for LSDA messages */
double        \LSDA_service_time;

/* service time for LSTA messages */
double        \LSTA_service_time;

/* service time for RTPC messages */
double        \RTPC_service_time;

/* service time for RTAC messages */
double        \RTAC_service_time;
```

```
/* service time for RTRC messages */
double      \RTRC_service_time;

/* service time for RSRT messages */
double      \RSRT_service_time;

/* service time for RTCC messages */
double      \RTCC_service_time;

/* service time for RCAT messages */
double      \RCAT_service_time;

/* timer 17 data type */
timer_type  \dv_timer17[SYS_LINK_LIMIT];

/* value for timer 17 */
double      \T17;

/* first activate timer17 */
int   \first_timer17;

/* last activate timer17 */
int   \last_timer17;

/* state variable for HMDC */
int   \dv_HMDC_state;

/* HMDC own SP restarting flag */
int   \dv_HMDC_own_SP_restarting_flag;

/* state variable for HMDT */
int   \dv_HMDT_state;

/* state variable for HMRT */
int   \dv_HMRT_state;

/* HMRT's List pointer for all destinations */
List *      \dv_HMRT_dest_list_ptr;

/* HMRT's table on all the links parameters */
HMRT_link_type    \dv_HMRT_link_table[SYS_LINK_LIMIT];

/* Signaling Link Management link table */
SLM_link_type     \SLM_link_table[SYS_LINK_LIMIT];

/* state variable for LSLA
 */
int   \dv_LSLA_state;

/* state variable for LSLR */
int   \dv_LSLR_state;

/* state variable for LSLD  */
int   \dv_LSLD_state;
```

```
/* log file handle for init state */
Log_Handle  \log_hdle_init;

/* log file handle for link startup */
Log_Handle  \log_hdle_link_startup;
```

APPENDIX C. MTP LEVEL 3 TEMPORARY VARIABLES

TEMPORARY VARIABLES BLOCK

```
Ici *               iciptr_input;
int                 ici_data;
Packet *            pkptr;
Packet*             pkptr_tx_pdu;
Packet*             pkptr_tx;
int                 pk_len_tx;
double              pk_tx_time;
Packet *            pkptr_rx;
Packet *            pkptr_rx_pdu;
int                 pk_len_rx;
int                 check;
int                 msg_code;
int                 service_indicator;

int                 adjSP;
int                 destSP;
int                 refSP;
int                 origSP;
int                 sig_link_selection;
int                 h0h1;
int                 link_num;
int                 sls;
List *              gdf_list_ptr;
char *              line_ptr;
List *              field_list_ptr;
node_type *         node_ptr;
node_type *         node_ptr2;
List *              new_route_list_ptr;
Route_Set *         routeset_ptr;
HMRT_route_type * new_route_ptr;
HMRT_route_type * route_ptr;

HMRT_sls_assign_type * sls_assign_ptr;
HMRT_sls_assign_type * new_sls_ptr;
HMRT_dest_type * dest_ptr;
HMRT_dest_type * new_dest_ptr;

int i;
int j;
int k;
int l;

int net_id;
int id;
int pc;
int type;
int adj_subnet_id;
int adj_user_id;
char str1[10];
char str2[10];
```

```
int route_has_SP;
int temp_net_id;
int temp_id;
int min_node_num;
```

APPENDIX D. MTP LEVEL 3 FUNCTION BLOCK

FUNCTION BLOCK

```
/* ==================================================================== */
/* Signaling System 7 (SS7), Message Transfer Part Level 3 (MTP3)       */
/* ITU-T Recommendation Q.704, Jul 1996                                 */
/* The definitions here are for these purposes                          */
/*   (1) Binary increment and decrement functions                       */
/*   (2) Timer start/stop functions                                     */
/*   (3) Packet Queuing functions for all internal messages             */
/* ==================================================================== */


/* ==================================================================== */
/* (1) Binary increment and decrement functions                         */
/* ==================================================================== */

/* define Binary increment */
int mtp3_increment(value, step, mod_number)
   int value;
   int step;
   int mod_number;
{
   int result;
   FIN (mtp3_increment(value, step, mod_number))

   result = (value + step) % mod_number;

   FRET (result);
}


/* define Binary decrement */
int mtp3_decrement(value, step, mod_number)
   int value;
   int step;
   int mod_number;
{
   int result;

   FIN (mtp3_decrement(value, step, mod_number));

   if ((value - step) < 0) {
      result = mod_number - (step - value);
   }
   else {
     result = value - step;
   }

   FRET (result);
}
```

```
/* ================================================================== */
/* (2) Timer start/stop functions                                      */
/* ================================================================== */

/* These functions emulate a "timer" by using OPNET's self interrupts */

/* Print a warning message */
void ss7_mtp3_timer_warn(msg)
   char*    msg;
{
   FIN (ss7_mtp3_timer_warn(msg));


   op_prg_odb_print_major ("Warning from SS7 timer functions:", msg, OPC_NIL);

   FOUT;
}


void mtp3_timer_start(timer)
   timer_type * timer;
{
   /* Start a timer, by queueing a self interrupt request.       */
   /* stop it first and restart if the timer is already running; */
   FIN (mtp3_timer_start (timer));

   if (timer->active) {
      if (op_ev_cancel(timer->ev) == OPC_COMPCODE_FAILURE) {
                  ss7_mtp3_timer_warn("Unable to cancel timer interrupt.");
      }
   }
   timer->ev = op_intrpt_schedule_self(op_sim_time() + timer->delay, timer->
   code);
   if (op_ev_valid(timer->ev) == OPC_FALSE) {
      ss7_mtp3_timer_warn("Unable to schedule timer interrupt.");
   }
   timer->active = 1;

   FOUT;
}

void mtp3_timer_stop(timer)
   timer_type * timer;
{
      /* Stop a timer by canceling its outstanding interrupt request.  */
      /* (Don't cancel the interrupt if it is the current one, as this */
      /* causes an error in the simulation kernel.)                    */
   FIN (mtp3__timer_stop(timer));

   if (timer->active && !(op_intrpt_type() == OPC_INTRPT_SELF &&
   op_intrpt_code() == timer->code)) {
      if (op_ev_cancel(timer->ev) == OPC_COMPCODE_FAILURE) {
         ss7_mtp3_timer_warn("Unable to cancel timer interrupt.");
      }
   }
      timer->active = 0;
      FOUT;
}

/* send an external message function */
void mtp3_send_external_msg(target_id, msg_code, msg_data, iciptr)
   Objid target_id;
   int msg_code;
   int msg_data;
   Ici * iciptr;
{
   char str1[10];

   FIN (mtp3_send_external_msg(target_id, msg_code, msg_data, iciptr));

   /* set the ICI data and install it to the current interrupt */
   op_ici_attr_set(iciptr, "data", msg_data);
   op_ici_install(iciptr);

   /* activate the interrupt to send out the message code & data value */
   op_intrpt_force_remote(msg_code, target_id);

   /* de-install the ICI so the they do not appear in future interrupts */
   op_ici_install(OPC_NIL);

  FOUT;
}

void discard_MSU(pkptr)
   Packet * pkptr;
{
   FIN(discard(pkptr));

   op_pk_destroy(pkptr);

   FOUT;
}


/* print a warning message */
void ss7_mtp3_warn (msg)
   char *   msg;
{

   /* Print a warning message */
   FIN (ss7_mtp3_warn (msg));

   op_prg_odb_print_major ("Warning from SS7 mtp3 functions:", msg, OPC_NIL);

   FOUT;
}
```

```
/* ============================================================== */
/* (3) Routing functions                                          */
/* ============================================================== */

/* pick a link for this destination based on sig Link Selection */
HMRT_sls_assign_type * sf_rt_HMRT_pick_a_link_for_tx(destSP, sls)
   int destSP;
   int sls;
{
   HMRT_dest_type * temp_dest_ptr;
   HMRT_dest_type * dest_ptr;
   HMRT_route_type * route_ptr;
   HMRT_sls_assign_type * sls_assign_ptr;
   HMRT_sls_assign_type * selected_sls_ptr;
   int accessible;
   int desired_link_available;
   int alternative_found;
   int i, j;

   FIN(sf_rt_HMRT_pick_a_link_for_tx(destSP, sls));

   selected_sls_ptr = OPC_NIL;
   accessible = FALSE;
   desired_link_available = TRUE;
   alternative_found = FALSE;

   for (i = 0; i < op_prg_list_size(dv_HMRT_dest_list_ptr); i++) {
      temp_dest_ptr = op_prg_list_access(dv_HMRT_dest_list_ptr, i);
      if ((temp_dest_ptr->point_code == destSP) && (temp_dest_ptr->
      dest_accessible)) {
         dest_ptr = temp_dest_ptr;
         accessible = TRUE;
      }
   }

   if (accessible) {

      /* find the link assigned to the sls and check availability */
      for (i = 0; i < op_prg_list_size(dest_ptr->route_list_ptr); i++) {
         route_ptr = op_prg_list_access(dest_ptr->route_list_ptr, i);
         for (j = 0; j < op_prg_list_size(route_ptr->sls_assign_list_ptr); j++){
            sls_assign_ptr = op_prg_list_access(route_ptr->sls_assign_list_ptr,
            j);
            if (sls_assign_ptr->sls_num == sls) {
               if (dv_HMRT_link_table[sls_assign_ptr->link_slc].link_available){
                  /* other conditions ??*/
                  selected_sls_ptr = sls_assign_ptr;
               }
               else {
                  desired_link_available = FALSE;
               }
            }
         }
      }
      /* pick the first available link in the route set */
      /* if the link assigned is not available         */
      if (!desired_link_available) {
         for (i = 0; i < op_prg_list_size(dest_ptr->route_list_ptr); i++) {
            route_ptr = op_prg_list_access(dest_ptr->route_list_ptr, i);
            for (j = 0; j < op_prg_list_size(route_ptr->sls_assign_list_ptr);
            j++) {
               sls_assign_ptr = op_prg_list_access(route_ptr->
               sls_assign_list_ptr, j);
               if ((dv_HMRT_link_table[sls_assign_ptr->link_slc].link_available)
               && (!alternative_found)) {
                  selected_sls_ptr = sls_assign_ptr;
                  alternative_found = TRUE;
               }
            }
         }
      }
   }

   FRET(selected_sls_ptr);
}

/* this function pick a link in the linkset adjoining the destSP */
HMRT_sls_assign_type * sf_rt_HMRT_pick_a_link_in_linkset_for_tx(destSP)
   int destSP;
{
   HMRT_dest_type * temp_dest_ptr;
   HMRT_dest_type * dest_ptr;
   HMRT_route_type * route_ptr;
   HMRT_sls_assign_type * sls_assign_ptr;
   HMRT_sls_assign_type * selected_sls_ptr;
   int accessible;
   int found;
   int i,j;

   FIN(sf_rt_HMRT_pick_a_link_in_linkset_for_tx(destSP));

   selected_sls_ptr = OPC_NIL;
   accessible = FALSE;
   found = FALSE;
   for (i = 0; i < op_prg_list_size(dv_HMRT_dest_list_ptr); i++) {
      temp_dest_ptr = op_prg_list_access(dv_HMRT_dest_list_ptr, i);
      if ((temp_dest_ptr->point_code == destSP) && (temp_dest_ptr->
      dest_accessible)) {
         dest_ptr = temp_dest_ptr;
         accessible = TRUE;
      }
   }

   if (accessible) {
      for (i = 0; i < op_prg_list_size(dest_ptr->route_list_ptr); i++) {
         route_ptr = op_prg_list_access(dest_ptr->route_list_ptr, i);
         for (j = 0; j < op_prg_list_size(route_ptr->sls_assign_list_ptr);
         j++) {
            sls_assign_ptr = op_prg_list_access(route_ptr->sls_assign_list_ptr,
            j);
            if ((dv_HMRT_link_table[sls_assign_ptr->link_slc].link_available)
            && (!found)) {
               selected_sls_ptr = sls_assign_ptr;
               found = TRUE;
            }
         }
      }
   }

   FRET(selected_sls_ptr);
}

/* this function picks the first available link found */
HMRT_sls_assign_type * sf_rt_HMRT_pick_any_link_for_tx()
{
   HMRT_dest_type * dest_ptr;
   HMRT_route_type * route_ptr;
   HMRT_sls_assign_type * sls_assign_ptr;
   HMRT_sls_assign_type * selected_sls_ptr;
   int found;
   int i,j,k;

   FIN(sf_rt_HMRT_pick_any_link_for_tx());

   selected_sls_ptr = OPC_NIL;
   found = FALSE;
   for (i = 0; i < op_prg_list_size(dv_HMRT_dest_list_ptr); i++) {
      dest_ptr = op_prg_list_access(dv_HMRT_dest_list_ptr, i);
      if (dest_ptr->dest_accessible) {
         for (j = 0; j < op_prg_list_size(dest_ptr->route_list_ptr); j++) {
            route_ptr = op_prg_list_access(dest_ptr->route_list_ptr, j);
            for (k = 0; k < op_prg_list_size(route_ptr->sls_assign_list_ptr);
            k++) {
               sls_assign_ptr = op_prg_list_access(route_ptr->
               sls_assign_list_ptr, k);
               if ((dv_HMRT_link_table[sls_assign_ptr->link_slc].link_available)
               && (!found)) {
                  selected_sls_ptr = sls_assign_ptr;
                  found = TRUE;
               }
            }
         }
      }
   }

   FRET(selected_sls_ptr);
}

/* this function finds the link assigned with the sls in the direct linkset */
/* with destSP and assumes it is available  */
HMRT_sls_assign_type * sf_rt_HMRT_pick_a_direct_link_for_tx(destSP, sls)
   int destSP;
   int sls;
{
   HMRT_dest_type * temp_dest_ptr;
   HMRT_dest_type * dest_ptr;
```

```
    HMRT_route_type * route_ptr;
    HMRT_sls_assign_type * sls_assign_ptr;
    HMRT_sls_assign_type * selected_sls_ptr;
    int accessible;
    int i,j;

    FIN(sf_rt_HMRT_pick_a_direct_link_for_tx(destSP));

    selected_sls_ptr = OPC_NIL;
    for (i = 0; i < op_prg_list_size(dv_HMRT_dest_list_ptr); i++) {
       temp_dest_ptr = op_prg_list_access(dv_HMRT_dest_list_ptr, i);
       if (temp_dest_ptr->point_code == destSP) {
          dest_ptr = temp_dest_ptr;
       }
    }

    for (i = 0; i < op_prg_list_size(dest_ptr->route_list_ptr); i++) {
       route_ptr = op_prg_list_access(dest_ptr->route_list_ptr, i);
       for (j = 0; j < op_prg_list_size(route_ptr->sls_assign_list_ptr); j++) {
          sls_assign_ptr = op_prg_list_access(route_ptr->sls_assign_list_ptr, j);
          if (sls_assign_ptr->sls_num == sls) {
             selected_sls_ptr = sls_assign_ptr;
          }
       }
    }

    FRET(selected_sls_ptr);
}

/* This function is not implemented */
void sf_rt_HMRT_update_routing_tables()
{
    int i;

    FIN(sf_rt_HMRT_update_routing_tables());

    i = 0;

    FOUT;
}

/* determine if any route towards destSP is available */
int sf_rt_HMRT_is_route_available(destSP)
    int destSP;
{
    int i;
    int num;
    int found;
    int status;
    HMRT_dest_type * dest_ptr;
    HMRT_route_type * route_ptr;


    FIN(sf_rt_HMRT_is_route_available(destSP));

    i = 0;
    found = FALSE;
    num = op_prg_list_size(dv_HMRT_dest_list_ptr);
    while ((!found) &&  (i < num)) {
       dest_ptr = op_prg_list_access(dv_HMRT_dest_list_ptr, i);
       if (dest_ptr->point_code == destSP) {
          found = TRUE;
       }
       i = i + 1;
    }


    i = 0;
    status = FALSE;
    num = op_prg_list_size(dest_ptr->route_list_ptr);
    while ((!status) && (i < num)) {
       route_ptr = op_prg_list_access(dest_ptr->route_list_ptr, i);
       status = route_ptr->route_available;
       i = i + 1;
    }

    FRET(status);
}

/* determine if destSP is an adjSP and is restarting */
int sf_rt_HMRT_is_DPC_adj_and_restarting(destSP)
    int destSP;
{
    int i;
    int found;
    int status;

    FIN(sf_rt_HMRT_is_DPC_adj_and_restarting(destSP));

    i = 0;
    found = FALSE;
    status = FALSE;
    while (!found && (i < link_total)) {
       if(dv_HMRT_link_table[i].SP_connected == destSP) {
          status = dv_HMRT_link_table[i].adjSP_restarting;
          found = TRUE;
       }
       i = i + 1;
    }

    FRET(status);
}
```

```
/* determine if destSP is congested */
int sf_rt_HMRT_is_DPC_congested(destSP)
   int destSP;
{
   int i;
   int num;
   int found;
   int status;
   HMRT_dest_type * dest_ptr;

   FIN(sf_rt_HMRT_is_DPC_congested(destSP));

   i = 0;
   found = FALSE;
   status = FALSE;
   num = op_prg_list_size(dv_HMRT_dest_list_ptr);
   while (!found &&  (i < num)) {
      dest_ptr = op_prg_list_access(dv_HMRT_dest_list_ptr, i);
      if (dest_ptr->point_code == destSP) {
         status = dest_ptr->dest_congested;
         found = TRUE;
      }
      i = i + 1;
   }


   FRET(status);
}

/* find the linkset status for links connected to adjSP */
int sf_rt_SLM_linkset_status(adjSP)
   int adjSP;
{
   int i;
   int found;
   int status;

   FIN(sf_rt_SLM_linkset_status(adjSP));

   found = FALSE;
   status = 0;
   i = 0;
   while (!found && (i< link_total)) {
      if (SLM_link_table[i].SP_connected == adjSP) {
         status = SLM_link_table[i].linkset_status;
         found = TRUE;
      }
      i = i + 1 ;
   }

   FRET(status);
}
```

```
/* set linkset_status to active for all links connected to adjSP */
void sf_rt_SLM_set_linkset_active(adjSP)
   int adjSP;

{
   int i;

   FIN(sf_rt_SLM_set_linkset_active(adjSP));

   for (i = 0; i < link_total; i++) {
      if (SLM_link_table[i].SP_connected == adjSP) {
         SLM_link_table[i].linkset_status = LINKSET_ACTIVE;
      }
   }

   FOUT;
}

/* set linkset_status to active for all links connected to adjSP */
void sf_rt_SLM_cancel_linkset_active(adjSP)
   int adjSP;

{
   int i;


   FIN(sf_rt_SLM_cancel_linkset_active(adjSP));


   for (i = 0; i < link_total; i++) {
      if (SLM_link_table[i].SP_connected == adjSP) {
         SLM_link_table[i].linkset_status = LINKSET_INACTIVE;
      }
   }


   FOUT;
}


/* ================================================================== */
/* (3) Packet Queuing functions for all internal messages            */
/* ================================================================== */

void SNM_CHANGEOVER_ORDER(pkptr)
   Packet * pkptr;
{
   Packet * newpkptr;
   double origSP;
   int last_accepted_FSN;

   FIN(SNM_CHANGEOVER_ORDER(pkptr));

   op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
   op_pk_fd_get(pkptr, FD_INDEX_7TH, &last_accepted_FSN);
   op_pk_destroy(pkptr);
```

```
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_TLAC_CHANGEOVER_ORDER, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    op_pk_fd_set(newpkptr, FD_INDEX_3RD, OPC_FIELD_TYPE_INTEGER,
    last_accepted_FSN, FD_LEN_FSN);
    if (op_subq_pk_insert(TLAC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(newpkptr);
    }

    FOUT;
}


void SNM_CHANGEOVER_ACK(pkptr)
    Packet * pkptr;
{
    Packet * newpkptr;
    double origSP;
    int last_accepted_FSN;

    FIN(SNM_CHANGEOVER_ACK(pkptr));

    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_fd_get(pkptr, FD_INDEX_7TH, &last_accepted_FSN);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_TCOC_CHANGEOVER_ACK, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    op_pk_fd_set(newpkptr, FD_INDEX_3RD, OPC_FIELD_TYPE_INTEGER,
    last_accepted_FSN, FD_LEN_FSN);
    if (op_subq_pk_insert(TCOC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(newpkptr);
    }

    FOUT;
}


void SNM_CHANGEBACK_DECLARATION(pkptr)
    Packet * pkptr;
{
    Packet * newpkptr;
    double origSP;

    FIN(SNM_CHANGEBACK_DECLARATION(pkptr));

    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_TCBC_CHANGEBACK_DECLARATION, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    if (op_subq_pk_insert(TCBC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(newpkptr);
    }

    FOUT;
}


void SNM_CHANGEBACK_ACK(pkptr)
    Packet * pkptr;
{
    Packet * newpkptr;
    double origSP;

    FIN(SNM_CHANGEBACK_ACK(pkptr));

    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_TCBC_CHANGEBACK_ACK, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    if (op_subq_pk_insert(TCBC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(newpkptr);
    }

    FOUT;
}


void SNM_EMERGENCY_CHANGEOVER_ORDER(pkptr)
    Packet * pkptr;
{
    Packet * newpkptr;
    double origSP;

    FIN(SNM_EMERGENCY_CHANGEOVER_ORDER(pkptr));

    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_TLAC_EMERGENCY_CHANGEOVER_ORDER, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    if (op_subq_pk_insert(TLAC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(newpkptr);
    }
    FOUT;
}
```

```
void SNM_EMERGENCY_CHANGEOVER_ACK(pkptr)
   Packet * pkptr;
{
   Packet * newpkptr;
   double origSP;

   FIN(SNM_EMERGENCY_CHANGEOVER_ACK(pkptr));

   op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
   op_pk_destroy(pkptr);
   newpkptr = op_pk_create(0);
   op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_HMDT_TCOC_EMERGENCY_CHANGEOVER_ACK, 16);
   op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
   FD_LEN_OPC);
   if (op_subq_pk_insert(TCOC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(newpkptr);
   }

   FOUT;
}


void SNM_TRANSFER_PROHIBITED(pkptr)
   Packet * pkptr;
{
   Packet * newpkptr;
   double origSP;
   double refSP;

   FIN(SNM_TRANSFER_PROHIBITED(pkptr));

   op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
   op_pk_fd_get(pkptr, FD_INDEX_7TH, &refSP);
   op_pk_destroy(pkptr);
   newpkptr = op_pk_create(0);
   op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_HMDT_RTPC_TRANSFER_PROHIBITED, 16);
   op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
   FD_LEN_OPC);
   op_pk_fd_set(newpkptr, FD_INDEX_3RD, OPC_FIELD_TYPE_INTEGER, refSP,
   FD_LEN_DPC);
   if (op_subq_pk_insert(RTPC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(newpkptr);
   }

   FOUT;
}
```

```
void SNM_TRANSFER_ALLOWED(pkptr)
   Packet * pkptr;
{
   Packet * newpkptr;
   double origSP;
   double refSP;

   FIN(SNM_TRANSFER_ALLOWED(pkptr));

   op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
   op_pk_fd_get(pkptr, FD_INDEX_7TH, &refSP);
   op_pk_destroy(pkptr);
   newpkptr = op_pk_create(0);
   op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_HMDT_RTAC_TRANSFER_ALLOWED, 16);
   op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
   FD_LEN_OPC);
   op_pk_fd_set(newpkptr, FD_INDEX_3RD, OPC_FIELD_TYPE_INTEGER, refSP,
   FD_LEN_DPC);
   if (op_subq_pk_insert(RTAC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(newpkptr);
   }

   FOUT;
}


void SNM_TRANSFER_RESTRICTED(pkptr)
   Packet * pkptr;
{
   Packet * newpkptr;
   double origSP;
   double refSP;

   FIN(SNM_TRANSFER_RESTRICTED(pkptr));

   op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
   op_pk_fd_get(pkptr, FD_INDEX_7TH, &refSP);
   op_pk_destroy(pkptr);
   newpkptr = op_pk_create(0);
   op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_HMDT_RTRC_TRANSFER_RESTRICTED, 16);
   op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
   FD_LEN_OPC);
   op_pk_fd_set(newpkptr, FD_INDEX_3RD, OPC_FIELD_TYPE_INTEGER, refSP,
   FD_LEN_DPC);
   if (op_subq_pk_insert(RTRC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(newpkptr);
   }

   FOUT;
}
```

```
void SNM_SIG_ROUTE_SET_TEST_PROHIBITED(pkptr)
   Packet * pkptr;
{
   Packet * newpkptr;
   double origSP;
   double refSP;

   FIN(SNM_SIG_ROUTE_SET_TEST_PROHIBITED(pkptr));

   op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
   op_pk_fd_get(pkptr, FD_INDEX_7TH, &refSP);
   op_pk_destroy(pkptr);
   newpkptr = op_pk_create(0);
   op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_HMDT_RSRT_SIG_ROUTE_SET_TEST_PROHIBITED, 16);
   op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
   FD_LEN_OPC);
   op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, refSP,
   FD_LEN_DPC);
   if (op_subq_pk_insert(RSRT_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(newpkptr);
   }

   FOUT;
}


void SNM_SIG_ROUTE_SET_TEST_RESTRICTED(pkptr)
   Packet * pkptr;
{
   Packet * newpkptr;
   double origSP;
   double refSP;

   FIN(SNM_SIG_ROUTE_SET_TEST_RESTRICTED(pkptr));

   op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
   op_pk_fd_get(pkptr, FD_INDEX_7TH, &refSP);
   op_pk_destroy(pkptr);
   newpkptr = op_pk_create(0);
   op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_HMDT_RSRT_SIG_ROUTE_SET_TEST_RESTRICTED, 16);
   op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
   FD_LEN_OPC);
   op_pk_fd_set(newpkptr, FD_INDEX_3RD, OPC_FIELD_TYPE_INTEGER, refSP,
   FD_LEN_DPC);
   if (op_subq_pk_insert(RSRT_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(newpkptr);
   }

   FOUT;
}
```

```
void SNM_LINK_INHIBIT(pkptr)
   Packet * pkptr;
{
   Packet * newpkptr;
   double origSP;

   FIN(SNM_LINK_INHIBIT(pkptr));

   op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
   op_pk_destroy(pkptr);
   newpkptr = op_pk_create(0);
   op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_HMDT_TLAC_LINK_INHIBIT, 16);
   op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
   FD_LEN_OPC);
   if (op_subq_pk_insert(TLAC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(newpkptr);
   }

   FOUT;
}


void SNM_LINK_UNINHIBIT(pkptr)
   Packet * pkptr;
{
   Packet * newpkptr;
   double origSP;

   FIN(SNM_LINK_UNINHIBIT(pkptr));

   op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
   op_pk_destroy(pkptr);
   newpkptr = op_pk_create(0);
   op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_HMDT_TLAC_LINK_UNINHIBIT, 16);
   op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
   FD_LEN_OPC);
   if (op_subq_pk_insert(TLAC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(newpkptr);
   }

   FOUT;
}


void SNM_LINK_INHIBIT_ACK(pkptr)
   Packet * pkptr;
{
   Packet * newpkptr;
   double origSP;

   FIN(SNM_LINK_INHIBIT_ACK(pkptr));

   op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
   op_pk_destroy(pkptr);
```

```
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_TLAC_LINK_INHIBIT_ACK, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    if (op_subq_pk_insert(TLAC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
       op_pk_destroy(newpkptr);
    }

    FOUT;
}


void SNM_LINK_UNINHIBIT_ACK(pkptr)
    Packet * pkptr;
{
    Packet * newpkptr;
    double origSP;

    FIN(SNM_LINK_UNINHIBIT_ACK(pkptr));

    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_TLAC_LINK_UNINHIBIT_ACK, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    if (op_subq_pk_insert(TLAC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
       op_pk_destroy(newpkptr);
    }

    FOUT;
}


void SNM_LINK_INHIBIT_DENIED(pkptr)
    Packet * pkptr;
{
    Packet * newpkptr;
    double origSP;

    FIN(SNM_LINK_INHIBIT_DENIED(pkptr));

    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_TLAC_LINK_INHIBIT_DENIED, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    if (op_subq_pk_insert(TLAC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
       op_pk_destroy(newpkptr);
    }

    FOUT;
```

```
}


void SNM_LINK_FORCE_UNINHIBIT(pkptr)
    Packet * pkptr;
{
    Packet * newpkptr;
    double origSP;

    FIN(SNM_LINK_FORCE_UNINHIBIT(pkptr));

    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_TLAC_LINK_FORCE_UNINHIBIT, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    if (op_subq_pk_insert(TLAC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(newpkptr);
    }

    FOUT;
}


void SNM_LINK_LOCAL_INHIBIT_TEST(pkptr)
    Packet * pkptr;
{
    Packet * newpkptr;
    double origSP;

    FIN(SNM_LINK_LOCAL_INHIBIT_TEST(pkptr));

    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_TLAC_LINK_LOCAL_INHIBIT_TEST, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    if (op_subq_pk_insert(TLAC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(newpkptr);
    }

    FOUT;
}


void SNM_LINK_REMOTE_INHIBIT_TEST(pkptr)
    Packet * pkptr;
{
    Packet * newpkptr;
    double origSP;

    FIN(SNM_LINK_REMOTE_INHIBIT_TEST(pkptr));
    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_TLAC_LINK_REMOTE_INHIBIT_TEST, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    if (op_subq_pk_insert(TLAC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
       op_pk_destroy(newpkptr);
    }

    FOUT;
}


void SNM_TRAFFIC_RESTART_ALLOWED(pkptr)
    Packet * pkptr;
{
    Packet * newpkptr;
    double origSP;

    FIN(SNM_TRAFFIC_RESTART_ALLOWED(pkptr));

    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_TPRC_TRAFFIC_RESTART_ALLOWED, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    if (op_subq_pk_insert(TPRC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
       op_pk_destroy(newpkptr);
    }

    FOUT;
}


void SNM_DATALINK_CONNECTION_ORDER(pkptr)
    Packet * pkptr;
{
    Packet * newpkptr;
    double origSP;
    int sig_datalink_id;

    FIN(SNM_DATALINK_CONNECTION_ORDER(pkptr));

    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_fd_get(pkptr, FD_INDEX_7TH, &sig_datalink_id);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_LSDA_DATALINK_CONNECTION_ORDER, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    op_pk_fd_set(newpkptr, FD_INDEX_3RD, OPC_FIELD_TYPE_INTEGER, sig_datalink_id,
    12);

    if (op_subq_pk_insert(LSDA_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(newpkptr);
    }

    FOUT;
}


void SNM_DATALINK_CONNECTION_SUCCESSFUL(pkptr)
    Packet * pkptr;
{
    Packet * newpkptr;
    double origSP;

    FIN(SNM_DATALINK_CONNECTION_SUCCESSFUL(pkptr));

    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_LSDA_DATALINK_CONNECTION_SUCCESSFUL, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    if (op_subq_pk_insert(LSDA_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(newpkptr);
    }

    FOUT;
}


void SNM_DATALINK_CONNECTION_NOT_SUCCESSFUL(pkptr)
    Packet * pkptr;
{
    Packet * newpkptr;
    double origSP;

    FIN(SNM_DATALINK_CONNECTION_NOT_SUCCESSFUL(pkptr));

    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_LSDA_DATALINK_CONNECTION_NOT_SUCCESSFUL, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    if (op_subq_pk_insert(LSDA_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(newpkptr);
    }

    FOUT;
}
```

```
void SNM_DATALINK_CONNECTION_NOT_POSSIBLE(pkptr)
   Packet * pkptr;
{
   Packet * newpkptr;
   double origSP;

   FIN(SNM_DATALINK_CONNECTION_NOT_POSSIBLE(pkptr));

   op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
   op_pk_destroy(pkptr);
   newpkptr = op_pk_create(0);
   op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_HMDT_LSDA_DATALINK_CONNECTION_NOT_POSSIBLE, 16);
   op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
   FD_LEN_OPC);
   if (op_subq_pk_insert(LSDA_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(newpkptr);
   }

   FOUT;
}


void SNM_TRANSFER_CONTROLLED(pkptr)
   Packet * pkptr;
{
   Packet * newpkptr;
   double origSP;
   double refSP;

   FIN(SNM_TRANSFER_CONTROLLED(pkptr));

   op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
   op_pk_fd_get(pkptr, FD_INDEX_7TH, &refSP);
   op_pk_destroy(pkptr);
   newpkptr = op_pk_create(0);
   op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_HMDT_RTCC_TRANSFER_CONTROLLED, 16);
   op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
   FD_LEN_OPC);
   op_pk_fd_set(newpkptr, FD_INDEX_3RD, OPC_FIELD_TYPE_INTEGER, refSP,
   FD_LEN_OPC);
   if (op_subq_pk_insert(RTCC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(newpkptr);
   }

   FOUT;
}


void SNM_SIG_ROUTE_SET_CONGESTION_TEST(pkptr)
   Packet * pkptr;
{
   Packet * newpkptr;
   double origSP;
```

```
    FIN(SNM_SIG_ROUTE_SET_CONGESTION_TEST(pkptr));

    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_RCAT_SIG_ROUTE_SET_CONGESTION_TEST, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    if (op_subq_pk_insert(RCAT_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
       op_pk_destroy(newpkptr);
    }

    FOUT;
}


void SNM_USER_PART_UNAVAILABLE(pkptr)
    Packet * pkptr;
{
    Packet * newpkptr;
    double origSP;
    double refSP;
    int user_part_id;
    int unavail_cause;

    FIN(SNM_USER_PART_UNAVAILABLE(pkptr));

    op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
    op_pk_fd_get(pkptr, FD_INDEX_7TH, &refSP);
    op_pk_fd_get(pkptr, FD_INDEX_9TH, &user_part_id);
    op_pk_fd_get(pkptr, FD_INDEX_10TH, &unavail_cause);
    op_pk_destroy(pkptr);
    newpkptr = op_pk_create(0);
    op_pk_fd_set(newpkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_HMDT_RTCC_USER_PART_AVAILABLE, 16);
    op_pk_fd_set(newpkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
    FD_LEN_OPC);
    op_pk_fd_set(newpkptr, FD_INDEX_3RD, OPC_FIELD_TYPE_INTEGER, refSP,
    FD_LEN_OPC);
    op_pk_fd_set(newpkptr, FD_INDEX_3RD, OPC_FIELD_TYPE_INTEGER, user_part_id,
    4);
    op_pk_fd_set(newpkptr, FD_INDEX_3RD, OPC_FIELD_TYPE_INTEGER, unavail_cause,
    4);
    if (op_subq_pk_insert(RTCC_BUFFER, newpkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
       op_pk_destroy(newpkptr);
    }

    FOUT;
}
```

```
/* internal packet queuing Messages */
void TPRC_LLSC_RESTART_BEGINS(adjSP)
   int adjSP;
{
   Packet * pkptr;


   FIN(TPRC_LLSC_RESTART_BEGINS(adjSP));


   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_TPRC_LLSC_RESTART_BEGINS, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, adjSP, FD_LEN_OPC);
   if (op_subq_pk_insert(LLSC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }


   FOUT;


void TPRC_LLSC_RESTART_ENDS(adjSP)
   int adjSP;
{
   Packet * pkptr;


   FIN(TPRC_LLSC_RESTART_ENDS(adjSP));


   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_TPRC_LLSC_RESTART_ENDS, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, adjSP, FD_LEN_OPC);
   if (op_subq_pk_insert(LLSC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }


   FOUT;
}


void LSAC_LLSC_LINK_ACTIVE(adjSP)
   int adjSP;
{
   Packet * pkptr;


   FIN(LSAC_LLSC_LINK_ACTIVE(adjSP));


   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSAC_LLSC_LINK_ACTIVE, 16);
```

```
    op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, adjSP, FD_LEN_OPC);
    if (op_subq_pk_insert(LLSC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
       op_pk_destroy(pkptr);
    }


    FOUT;
}

void LSAC_LLSC_LINK_INACTIVE(adjSP)
    int adjSP;
{
    Packet * pkptr;


    FIN(LSAC_LLSC_LINK_INACTIVE(adjSP));


    pkptr = op_pk_create(0);
    op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_LSAC_LLSC_LINK_INACTIVE, 16);
    op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, adjSP, FD_LEN_OPC);
    if (op_subq_pk_insert(LLSC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
       op_pk_destroy(pkptr);
    }


    FOUT;
}

void LSAC_LLSC_LINK_FAILED(adjSP)
    int adjSP;
{
    Packet * pkptr;

    FIN(LSAC_LLSC_LINK_FAILED(adjSP));

    pkptr = op_pk_create(0);
    op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_LSAC_LLSC_LINK_FAILED, 16);
    op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, adjSP, FD_LEN_OPC);
    if (op_subq_pk_insert(LLSC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
       op_pk_destroy(pkptr);
    }


    FOUT;
}


void LSAC_LLSC_ACTIVATE_ANOTHER_LINK(adjSP)
    int adjSP;
{
    Packet * pkptr;

    FIN(LSAC_LLSC_ACTIVATE_ANOTHER_LINK(adjSP));
    pkptr = op_pk_create(0);
    op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_LSAC_LLSC_ACTIVATE_ANOTHER_LINK, 16);
    op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, adjSP, FD_LEN_OPC);
    if (op_subq_pk_insert(LLSC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(pkptr);
    }


    FOUT;
}


void TSRC_LLSC_EMERGENCY(adjSP)
    int adjSP;
{
    Packet * pkptr;

    FIN(TSRC_LLSC_EMERGENCY(adjSP));

    pkptr = op_pk_create(0);
    op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_TSRC_LLSC_EMERGENCY, 16);
    op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, adjSP, FD_LEN_OPC);
    if (op_subq_pk_insert(LLSC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(pkptr);
    }

    FOUT;
}


void TSRC_LLSC_EMERGENCY_CEASES(adjSP)
    int adjSP;
{
    Packet * pkptr;

    FIN(TSRC_LLSC_EMERGENCY_CEASES(adjSP));

    pkptr = op_pk_create(0);
    op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_TSRC_LLSC_EMERGENCY_CEASES, 16);
    op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, adjSP, FD_LEN_OPC);
    if (op_subq_pk_insert(LLSC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(pkptr);
    }


    FOUT;
}
```

```
void TLAC_LSAC_CONTINUE(link_num)
    int link_num;
{
    Packet * pkptr;

    FIN(TLAC_LSAC_CONTINUE(link_num));

    pkptr = op_pk_create(0);
    op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_TLAC_LSAC_CONTINUE, 16);
    op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
    if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(pkptr);
    }


    FOUT;
}


void TLAC_LSAC_FLUSH_BUFFERS(link_num)
    int link_num;
{
    Packet * pkptr;

    FIN(TLAC_LSAC_FLUSH_BUFFERS(link_num));

    pkptr = op_pk_create(0);
    op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_TLAC_LSAC_FLUSH_BUFFERS, 16);
    op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
    if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(pkptr);
    }

    FOUT;
}


void TLAC_LSAC_CHANGEOVER_ORDER_RECEIVED(link_num)
    int link_num;
{
    Packet * pkptr;

    FIN(TLAC_LSAC_CHANGEOVER_ORDER_RECEIVED(link_num));

    pkptr = op_pk_create(0);
    op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
    MSG_TLAC_LSAC_CHANGEOVER_ORDER_RECEIVED, 16);
    op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
    if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
        op_pk_destroy(pkptr);
    }

    FOUT;
}
```

```
void TCOC_LSAC_STM_READY(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(TCOC_LSAC_STM_READY(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_TCOC_LSAC_STM_READY, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void TCOC_LSAC_STOP_L2(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(TCOC_LSAC_STOP_L2(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_TCOC_LSAC_STOP_L2, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void LLSC_LSAC_ACTIVATE_LINK(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LLSC_LSAC_ACTIVATE_LINK(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LLSC_LSAC_ACTIVATE_LINK, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}
```

```
void LLSC_LSAC_DEACTIVATE_LINK(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LLSC_LSAC_DEACTIVATE_LINK(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LLSC_LSAC_DEACTIVATE_LINK, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void LLSC_LSAC_EMERGENCY(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LLSC_LSAC_EMERGENCY(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LLSC_LSAC_EMERGENCY, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void LLSC_LSAC_EMERGENCY_CEASES(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LLSC_LSAC_EMERGENCY_CEASES(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LLSC_LSAC_EMERGENCY_CEASES, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}
```

```
void LLSC_LSAC_RESTART_ENDS(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LLSC_LSAC_RESTART_ENDS(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LLSC_LSAC_RESTART_ENDS, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void LSLA_LSAC_START_SIG_LINK(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSLA_LSAC_START_SIG_LINK(link_num));

  pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSLA_LSAC_START_SIG_LINK, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void LSLA_LSAC_ACTIVATION_UNSUCCESSFUL(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSLA_LSAC_ACTIVATION_UNSUCCESSFUL(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSLA_LSAC_ACTIVATION_UNSUCCESSFUL, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}
```

```
void LSLR_LSAC_START_SIG_LINK(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSLR_LSAC_START_SIG_LINK(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSLR_LSAC_START_SIG_LINK, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void LSLR_LSAC_RESTORATION_UNSUCCESSFUL(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSLR_LSAC_RESTORATION_UNSUCCESSFUL(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSLR_LSAC_RESTORATION_UNSUCCESSFUL, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void SLTC_LSAC_SLT_SUCCESSFUL(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(SLTC_LSAC_SLT_SUCCESSFUL(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_SLTC_LSAC_SLT_SUCCESSFUL, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}
```

```
void SLTC_LSAC_SLT_FAILED(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(SLTC_LSAC_SLT_FAILED(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_SLTC_LSAC_SLT_FAILED, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}

void LSC_LSAC_IN_SERVICE(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSC_LSAC_IN_SERVICE(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSC_LSAC_IN_SERVICE, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}

void LSC_LSAC_OUT_SERVICE(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSC_LSAC_OUT_SERVICE(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSC_LSAC_OUT_SERVICE, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }
   FOUT;
}


void LSC_LSAC_REMOTE_PRO_RECOVERED(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSC_LSAC_REMOTE_PRO_RECOVERED(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSC_LSAC_REMOTE_PRO_RECOVERED, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void LSC_LSAC_REMOTE_PRO_OUTAGE(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSC_LSAC_REMOTE_PRO_OUTAGE(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSC_LSAC_REMOTE_PRO_OUTAGE, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void LSAC_LSLA_START_ACTIVATION(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSAC_LSLA_START_ACTIVATION(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSAC_LSLA_START_ACTIVATION, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSLA_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }
   FOUT;
}

void LSAC_LSLA_RESTART_ACTIVATION(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSAC_LSLA_RESTART_ACTIVATION(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSAC_LSLA_RESTART_ACTIVATION, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSLA_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void LSAC_LSLA_DEACTIVATE_LINK(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSAC_LSLA_DEACTIVATE_LINK(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSAC_LSLA_DEACTIVATE_LINK, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSLA_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void LSAC_LSLR_START_RESTORATION(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSAC_LSLR_START_RESTORATION(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSAC_LSLR_START_RESTORATION, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSLR_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }
   FOUT;
}


void LSAC_LSLR_RESTART_RESTORATION(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSAC_LSLR_RESTART_RESTORATION(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSAC_LSLR_RESTART_RESTORATION, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSLR_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void LSAC_LSLD_DEACTIVATE_LINK(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSAC_LSLD_DEACTIVATE_LINK(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSAC_LSLD_DEACTIVATE_LINK, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(LSLD_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void LSAC_TLAC_REMOTE_PRO_OUTAGE(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSAC_TLAC_REMOTE_PRO_OUTAGE(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSAC_TLAC_REMOTE_PRO_OUTAGE, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(TLAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }
   FOUT;
}

void LSAC_TLAC_REMOTE_PRO_RECOVERED(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSAC_TLAC_REMOTE_PRO_RECOVERED(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSAC_TLAC_REMOTE_PRO_RECOVERED, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(TLAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void LSAC_TLAC_LINK_IN_SERVICE(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSAC_TLAC_LINK_IN_SERVICE(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSAC_TLAC_LINK_IN_SERVICE, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(TLAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void LSAC_TLAC_LINK_FAILED(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSAC_TLAC_LINK_FAILED(link_num));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_LSAC_TLAC_LINK_FAILED, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num, 16);
   if (op_subq_pk_insert(TLAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }
   FOUT;
}


void HMRT_TSFC_MSG_FOR_CONGESTED_DEST(origSP, destSP)
   int origSP;
   int destSP;
{
   Packet * pkptr;

   FIN(HMRT_TSFC_MSG_FOR_CONGESTED_DEST(origSP, destSP));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_HMRT_TSFC_MSG_FOR_CONGESTED_DEST, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
   FD_LEN_OPC);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, destSP,
   FD_LEN_DPC);
   if (op_subq_pk_insert(TSFC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void HMRT_RTPC_MSG_RECEIVED_FOR_INACCESSIBLE_SP(origSP, destSP)
   int origSP;
   int destSP;
{
   Packet * pkptr;

   FIN(HMRT_RTPC_MSG_RECEIVED_FOR_INACCESSIBLE_SP(origSP, destSP));

   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_HMRT_RTPC_MSG_RECEIVED_FOR_INACCESSIBLE_SP, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, origSP,
   FD_LEN_OPC);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, destSP,
   FD_LEN_DPC);
   if (op_subq_pk_insert(RTPC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}


void HMRT_MGMT_MSG_RECEIVED_FOR_INACCESSIBLE_SP(origSP, destSP)
   int origSP;
   int destSP;
{
   int i;
   FIN(HMRT_MGMT_MSG_RECEIVED_FOR_INACCESSIBLE_SP(origSP, destSP));

   i = 0;

   FOUT;
}


/* this function should send message to SLTC */
/* since SLTC is not implemented, a reply is simulated for loop completion */

void LSAC_SLTC_START_SLT(link_num)
   int link_num;
{
   Packet * pkptr;

   FIN(LSAC_SLTC_START_SLT(link_num));
   pkptr = op_pk_create(0);
   op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER,
   MSG_SLTC_LSAC_SLT_SUCCESSFUL, 16);
   op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, link_num,
   FD_LEN_OPC);
   if (op_subq_pk_insert(LSAC_BUFFER, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK) {
      op_pk_destroy(pkptr);
   }

   FOUT;
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. EXECUTIVES FOR "INIT" STATE

EXECUTIVES FOR "INIT" STATE

```
log_hdle_init = op_prg_log_handle_create(OpC_Log_Category_Configuration, "MTP3",
"Init", 1000);
log_hdle_link_startup = op_prg_log_handle_create(OpC_Log_Category_Configuration,
"MTP3", "Link Startup", 2000);


/* get queue module's own object id and that of MTP3 */
own_objid = op_id_self();

/* get assigned values for all the model attributes    */
op_ima_obj_attr_get(op_topo_parent(own_objid), "user id", &own_user_id);
op_ima_obj_attr_get(op_topo_parent(op_topo_parent(own_objid)), "user id",
&own_subnet_user_id);
op_ima_obj_attr_get(own_objid, "HMDC_MSU_service_time", &HMDC_MSU_service_time);
op_ima_obj_attr_get(own_objid, "HMDT_MSU_service_time", &HMDT_MSU_service_time);
op_ima_obj_attr_get(own_objid, "HMRT_MSU_service_time", &HMRT_MSU_service_time);
op_ima_obj_attr_get(own_objid, "HMDC_service_time", &HMDC_service_time);
op_ima_obj_attr_get(own_objid, "HMDT_service_time", &HMDT_service_time);
op_ima_obj_attr_get(own_objid, "HMRT_service_time", &HMRT_service_time);
op_ima_obj_attr_get(own_objid, "TLAC_service_time", &TLAC_service_time);
op_ima_obj_attr_get(own_objid, "TSRC_service_time", &TSRC_service_time);
op_ima_obj_attr_get(own_objid, "TSFC_service_time", &TSFC_service_time);
op_ima_obj_attr_get(own_objid, "TPRC_service_time", &TPRC_service_time);
op_ima_obj_attr_get(own_objid, "TCOC_service_time", &TCOC_service_time);
op_ima_obj_attr_get(own_objid, "TCBC_service_time", &TCBC_service_time);
op_ima_obj_attr_get(own_objid, "TFRC_service_time", &TFRC_service_time);
op_ima_obj_attr_get(own_objid, "TCRC_service_time", &TCRC_service_time);
op_ima_obj_attr_get(own_objid, "LLSC_service_time", &LLSC_service_time);
op_ima_obj_attr_get(own_objid, "LSAC_service_time", &LSAC_service_time);
op_ima_obj_attr_get(own_objid, "LSLA_service_time", &LSLA_service_time);
op_ima_obj_attr_get(own_objid, "LSLR_service_time", &LSLR_service_time);
op_ima_obj_attr_get(own_objid, "LSLD_service_time", &LSLD_service_time);
op_ima_obj_attr_get(own_objid, "LSTA_service_time", &LSTA_service_time);
op_ima_obj_attr_get(own_objid, "LSDA_service_time", &LSDA_service_time);
op_ima_obj_attr_get(own_objid, "RTPC_service_time", &RTPC_service_time);
op_ima_obj_attr_get(own_objid, "RTAC_service_time", &RTAC_service_time);
op_ima_obj_attr_get(own_objid, "RTRC_service_time", &RTRC_service_time);
op_ima_obj_attr_get(own_objid, "RSRT_service_time", &RSRT_service_time);
op_ima_obj_attr_get(own_objid, "RTCC_service_time", &RTCC_service_time);
op_ima_obj_attr_get(own_objid, "RCAT_service_time", &RCAT_service_time);
op_ima_obj_attr_get(own_objid, "T17", &T17);


/* convert service time from milliseconds to seconds */
HMDC_MSU_service_time = HMDC_MSU_service_time/1000;
HMDT_MSU_service_time = HMDT_MSU_service_time/1000;
HMRT_MSU_service_time = HMRT_MSU_service_time/1000;
HMDC_service_time = HMDC_service_time/1000;
HMDT_service_time = HMDT_service_time/1000;
HMRT_service_time = HMRT_service_time/1000;
```

```
TLAC_service_time = TLAC_service_time/1000;
TSRC_service_time = TSRC_service_time/1000;
TSFC_service_time = TSFC_service_time/1000;
TPRC_service_time = TPRC_service_time/1000;
TCOC_service_time = TCOC_service_time/1000;
TCBC_service_time = TCBC_service_time/1000;
TFRC_service_time = TFRC_service_time/1000;
TCRC_service_time = TCRC_service_time/1000;
LLSC_service_time = LLSC_service_time/1000;
LSAC_service_time = LSAC_service_time/1000;
LSLA_service_time = LSLA_service_time/1000;
LSLR_service_time = LSLR_service_time/1000;
LSLD_service_time = LSLD_service_time/1000;
LSTA_service_time = LSTA_service_time/1000;
LSDA_service_time = LSDA_service_time/1000;
RTPC_service_time = RTPC_service_time/1000;
RTAC_service_time = RTAC_service_time/1000;
RTRC_service_time = RTRC_service_time/1000;
RSRT_service_time = RSRT_service_time/1000;
RTCC_service_time = RTCC_service_time/1000;
RCAT_service_time = RCAT_service_time/1000;

/* start message processing for all functional blocks */
op_intrpt_schedule_self(op_sim_time() + HMDC_MSU_service_time, HMDC_MSU);
op_intrpt_schedule_self(op_sim_time() + HMDT_MSU_service_time, HMDT_MSU);
op_intrpt_schedule_self(op_sim_time() + HMRT_MSU_service_time, HMRT_MSU);
op_intrpt_schedule_self(op_sim_time() + HMDC_service_time, HMDC_MSG);
op_intrpt_schedule_self(op_sim_time() + HMDT_service_time, HMDT_MSG);
op_intrpt_schedule_self(op_sim_time() + HMRT_service_time, HMRT_MSG);
op_intrpt_schedule_self(op_sim_time() + TLAC_service_time, TLAC_MSG);
op_intrpt_schedule_self(op_sim_time() + TSRC_service_time, TSRC_MSG);
op_intrpt_schedule_self(op_sim_time() + TSFC_service_time, TSFC_MSG);
op_intrpt_schedule_self(op_sim_time() + TPRC_service_time, TPRC_MSG);
op_intrpt_schedule_self(op_sim_time() + TCOC_service_time, TCOC_MSG);
op_intrpt_schedule_self(op_sim_time() + TCBC_service_time, TCBC_MSG);
op_intrpt_schedule_self(op_sim_time() + TFRC_service_time, TFRC_MSG);
op_intrpt_schedule_self(op_sim_time() + TCRC_service_time, TCRC_MSG);
op_intrpt_schedule_self(op_sim_time() + LLSC_service_time, LLSC_MSG);
op_intrpt_schedule_self(op_sim_time() + LSAC_service_time, LSAC_MSG);
op_intrpt_schedule_self(op_sim_time() + LSLA_service_time, LSLA_MSG);
op_intrpt_schedule_self(op_sim_time() + LSLR_service_time, LSLR_MSG);
op_intrpt_schedule_self(op_sim_time() + LSLD_service_time, LSLD_MSG);
op_intrpt_schedule_self(op_sim_time() + LSDA_service_time, LSDA_MSG);
op_intrpt_schedule_self(op_sim_time() + LSTA_service_time, LSTA_MSG);
op_intrpt_schedule_self(op_sim_time() + RTPC_service_time, RTPC_MSG);
op_intrpt_schedule_self(op_sim_time() + RTAC_service_time, RTAC_MSG);
op_intrpt_schedule_self(op_sim_time() + RTRC_service_time, RTRC_MSG);
op_intrpt_schedule_self(op_sim_time() + RSRT_service_time, RSRT_MSG);
op_intrpt_schedule_self(op_sim_time() + RTCC_service_time, RTCC_MSG);
op_intrpt_schedule_self(op_sim_time() + RCAT_service_time, RCAT_MSG);
```

```
/* init Ici's */
iciptr_external = op_ici_create("ss7_ici_fmt");

/* init timer's */
first_timer17 = 0;
last_timer17 = 0;
for (i = 0; i < SYS_LINK_LIMIT; i++) {
   dv_timer17[i].code = T17_EXPIRE;
   dv_timer17[i].delay = T17;
}

/* initialize the states of all the functional blocks */
dv_HMDC_state = STATE_HMDC_IDLE;
dv_HMDT_state = STATE_HMDT_IDLE;
dv_HMRT_state = STATE_HMRT_IDLE;
dv_LSLA_state = STATE_LSLA_IDLE;
dv_LSLR_state = STATE_LSLR_IDLE;
dv_LSLD_state = STATE_LSLD_IDLE;

dv_HMDC_own_SP_restarting_flag = FALSE;


/*============================================================ */
/* Load the Node Table from "node.gdf"                        */
/* and determine own point code and type                      */
/*============================================================*/

gdf_list_ptr = op_prg_gdf_read("node");
node_table_ptr = op_prg_list_create();
node_total = op_prg_list_size(gdf_list_ptr);

/* loop through the table and store the values */
for (i = 0; i < node_total; i++) {

   node_ptr = op_prg_mem_alloc(sizeof(node_type));

   field_list_ptr = op_prg_str_decomp(op_prg_list_access(gdf_list_ptr, i),
   ",");

   node_ptr->subnet_user_id = atoi(op_prg_list_access(field_list_ptr, 0));
   node_ptr->user_id = atoi(op_prg_list_access(field_list_ptr, 1));
   node_ptr->point_code = atoi(op_prg_list_access(field_list_ptr, 2));
   node_ptr->type = atoi(op_prg_list_access(field_list_ptr, 3));

   if ((node_ptr->subnet_user_id == own_subnet_user_id) && (node_ptr->
   user_id == own_user_id)) {
      own_PC = node_ptr->point_code;
      own_SP_type = node_ptr->type;
   }

   op_prg_list_insert(node_table_ptr, node_ptr, OPC_LISTPOS_TAIL);
   op_prg_list_free(field_list_ptr); op_prg_mem_free(field_list_ptr);
}

/* deallocate the gdf list and list contents */
op_prg_list_free(gdf_list_ptr); op_prg_mem_free(gdf_list_ptr);
```

```
/* log the node table created */
op_prg_log_entry_write(log_hdle_init, "Own Subnet user id: %d\nOwn user id:
%d\nOwn Point Code: %d\nOwn type: %d\n", own_subnet_user_id, own_user_id,
own_PC, own_SP_type);
for (i = 0; i < op_prg_list_size(node_table_ptr); i++) {
   node_ptr = op_prg_list_access(node_table_ptr, i);
   op_prg_log_entry_write(log_hdle_init, "Node Table Entry:\n
   subnet_user_id[%d]\n   user_id [%d]\n   point_code[%d]\n   type[%d]\n",
   node_ptr->subnet_user_id, node_ptr->user_id, node_ptr->point_code,
   node_ptr->type);
}


/*============================================================ */
/* Load the link Table from "link.gdf"                          */
/* and determine number of links connected to ownself          */
/*=============================================================*/

gdf_list_ptr = op_prg_gdf_read("link");
i = 0; j = 0;
k = op_prg_list_size(gdf_list_ptr);
while (i < k) {
   line_ptr = op_prg_list_access(gdf_list_ptr, i);
   field_list_ptr = op_prg_str_decomp(line_ptr, ",");
   if (atoi(op_prg_list_access(field_list_ptr, 0)) == own_PC) {

      /* extract the values of the adjacent node id into the row */
      master_link_table[j].connected = TRUE;
      master_link_table[j].link_slc = j;
      master_link_table[j].stream_num = atoi(op_prg_list_access(field_list_ptr,
      1));
      master_link_table[j].SP_connected =
      atoi(op_prg_list_access(field_list_ptr, 2));
      j = j + 1;
   }

   i = i + 1;

   /* deallocate the field list contents and list */
   op_prg_list_free(field_list_ptr); op_prg_mem_free(field_list_ptr);
}

link_total = j;

for (k = j; k < SYS_LINK_LIMIT; k++) {
    master_link_table[j].connected = FALSE;
    master_link_table[j].link_slc = 0;
    master_link_table[j].stream_num = 0;
    master_link_table[j].SP_connected = 0;
}

/* deallocate the gdf list and list contents */
op_prg_list_free(gdf_list_ptr); op_prg_mem_free(gdf_list_ptr);
```

```
/* log the link table */
for (i = 0; i < link_total; i++) {
   op_prg_log_entry_write(log_hdle_init, "Master Link Table Entry:\n
   connected[%d]\n   link_slc[%d]\n   stream_num[%d]\n   SP_connected[%d]\n",
   master_link_table[i].connected, master_link_table[i].link_slc,
   master_link_table[i].stream_num, master_link_table[i].SP_connected);
}


/*============================================================ */
/* determine HMRT destination list                             */
/*============================================================*/

dv_HMRT_dest_list_ptr = op_prg_list_create();
network_ptr = op_rte_topo_from_userids();
for (i = 0; i < node_total; i++) {
   node_ptr = op_prg_list_access(node_table_ptr, i);
   net_id = node_ptr->subnet_user_id;
   id = node_ptr->user_id;
   pc = node_ptr->point_code;
   type = node_ptr->type;
   if ( pc != own_PC) {
      new_dest_ptr = op_prg_mem_alloc (sizeof (HMRT_dest_type));
      new_dest_ptr->subnet_user_id = net_id;
      new_dest_ptr->user_id = id;
      new_dest_ptr->point_code = pc;
      new_dest_ptr->node_type = type;
      new_dest_ptr->dest_accessible = TRUE;
      new_dest_ptr->dest_congested = FALSE;
      new_dest_ptr->route_list_ptr = op_prg_list_create();
      routeset_ptr = op_rte_routeset_create(network_ptr,
      own_subnet_user_id, own_user_id, net_id, id, SYS_MAX_HOPS);

      /* find the length of the shortest route */
      min_node_num = SYS_MAX_HOPS + 1;
      for (j = 0; j < op_rte_routeset_num_routes(routeset_ptr); j++) {
         k = op_rte_route_num_nodes(op_rte_routeset_route_n(routeset_ptr,
         j));
         if (k < min_node_num) {
            min_node_num = k;
         }
      }

      for (j = 0; j < op_rte_routeset_num_routes(routeset_ptr); j++) {

         if (op_rte_route_num_nodes(op_rte_routeset_route_n(routeset_ptr,
         j)) == min_node_num) {

            /* check that the route do not have a SP except for source and
            dest */
            route_has_SP = FALSE;
            for (k = 1; k <
            op_rte_route_num_nodes(op_rte_routeset_route_n(routeset_ptr,
            j)) - 1); k++) {
               op_rte_route_node_n(op_rte_routeset_route_n(routeset_ptr,
               j), k, &temp_net_id, &temp_id);
```

```
                for (l = 0; l < node_total; l++) {
                   node_ptr2 = op_prg_list_access(node_table_ptr, l);
                   if ((node_ptr2->subnet_user_id == temp_net_id) &&
                  (node_ptr2->user_id == temp_id) && (node_ptr2->type ==
                   SP)) {
                      route_has_SP = TRUE;
                   }
                }
             }

             /* use this route if it does not has a SP except for source
             and dest */
             if (!route_has_SP) {
                new_route_ptr = op_prg_mem_alloc (sizeof
               (HMRT_route_type));
                op_rte_route_node_n(op_rte_routeset_route_n(routeset_ptr,
                j), 1, &adj_subnet_id, &adj_user_id);

                /* find the point code of next SP to route to */
                for (k = 0; k < node_total; k++) {
                   node_ptr2 = op_prg_list_access(node_table_ptr, k);
                   if ((node_ptr2->subnet_user_id == adj_subnet_id) &&
                      (node_ptr2->user_id == adj_user_id)) {
                      new_route_ptr->nextSP = node_ptr2->point_code;
                   }
                }
                new_route_ptr->route_available = TRUE;
                new_route_ptr->sls_assign_list_ptr = op_prg_list_create();

                op_prg_list_insert(new_dest_ptr->route_list_ptr,
                new_route_ptr, OPC_LISTPOS_TAIL);
             }
          }
       }
       op_prg_list_insert(dv_HMRT_dest_list_ptr, new_dest_ptr,
       OPC_LISTPOS_TAIL);
    }
}

/* fill up the sls assignment */
for (i = 0; i < op_prg_list_size(dv_HMRT_dest_list_ptr); i++) {
   dest_ptr = op_prg_list_access(dv_HMRT_dest_list_ptr, i);
   sls = 0;
   while (sls < SYS_LINK_LIMIT) {
      for (j = 0; j < op_prg_list_size(dest_ptr->route_list_ptr); j++) {
         route_ptr = op_prg_list_access(dest_ptr->route_list_ptr, j);

         /* assign a sls to every link in the route     */
         /* repeat assignment till all sls are assigned */
         for (k = 0; k < link_total; k++) {
            if ((master_link_table[k].SP_connected == route_ptr->nextSP)
               && (sls < SYS_LINK_LIMIT)) {
               new_sls_ptr = op_prg_mem_alloc (sizeof
               HMRT_sls_assign_type));
               new_sls_ptr->sls_num = sls;
               new_sls_ptr->link_slc = master_link_table[k].link_slc;
```

```
                op_prg_list_insert(route_ptr->sls_assign_list_ptr,
                new_sls_ptr, OPC_LISTPOS_TAIL);
                sls = sls + 1;
            }
        }
    }
  }
}


/* log the dest list */
for (i = 0; i < op_prg_list_size(dv_HMRT_dest_list_ptr); i++) {
   dest_ptr = op_prg_list_access(dv_HMRT_dest_list_ptr, i);
   op_prg_log_entry_write(log_hdle_init, "HMRT dest Entry:\n
   subnet_user_id[%d]\n   user_id [%d]\n   point_code[%d]\n
   node_type[%d]\n   route_num[%d]\n", dest_ptr->subnet_user_id,
   dest_ptr->user_id, dest_ptr->point_code, dest_ptr->node_type,
   op_prg_list_size(dest_ptr->route_list_ptr));
   for (j = 0; j < op_prg_list_size(dest_ptr->route_list_ptr); j++) {
      route_ptr = op_prg_list_access(dest_ptr->route_list_ptr, j);
      op_prg_log_entry_write(log_hdle_init, "HMRT route Entry:\n
      nextSP[%d]\n   sls_assign_num[%d]\n", route_ptr->nextSP,
      op_prg_list_size(route_ptr->sls_assign_list_ptr));
      for (k = 0; k < op_prg_list_size(route_ptr->sls_assign_list_ptr);
      k++) {
         sls_assign_ptr = op_prg_list_access(route_ptr->
         sls_assign_list_ptr, k);
         op_prg_log_entry_write(log_hdle_init, "HMRT sls Entry:\n
         sls_num[%d]\n   link_slc[%d]\n", sls_assign_ptr->sls_num,
         sls_assign_ptr->link_slc);
      }
   }
}


/*============================================================ */
/* create dv_HMRT_link_table and SLM_link_table using master    */
/* Link table                                                   */
/*============================================================*/

for (i = 0; i < SYS_LINK_LIMIT; i++) {
   if (master_link_table[i].connected) {
      dv_HMRT_link_table[i].connected = TRUE;
      dv_HMRT_link_table[i].link_slc = master_link_table[i].link_slc;
      dv_HMRT_link_table[i].stream_num = master_link_table[i].stream_num;
      dv_HMRT_link_table[i].SP_connected =
      master_link_table[i].SP_connected;
      dv_HMRT_link_table[i].adjSP_restarting = FALSE;
      dv_HMRT_link_table[i].link_available = TRUE;
   }
   else {
      dv_HMRT_link_table[i].connected = FALSE;
      dv_HMRT_link_table[i].link_slc = 0;
      dv_HMRT_link_table[i].stream_num = 0;
      dv_HMRT_link_table[i].SP_connected = 0;
      dv_HMRT_link_table[i].adjSP_restarting = FALSE;
```

```
         dv_HMRT_link_table[i].link_available = TRUE;
      }
   }

   for (i = 0; i < SYS_LINK_LIMIT; i++) {
      if (master_link_table[i].connected) {
         SLM_link_table[i].connected = TRUE;
         SLM_link_table[i].link_slc = master_link_table[i].link_slc;
         SLM_link_table[i].stream_num = master_link_table[i].stream_num;
         SLM_link_table[i].SP_connected = master_link_table[i].SP_connected;
         _itoa(i, str1, 10);
         strcpy(str2, "MTP2_");
         strcat(str2, str1);
         SLM_link_table[i].link_objid = op_id_from_name(op_topo_parent(own_objid),
OPC_OBJTYPE_QUEUE, str2);
         SLM_link_table[i].linkset_status = LINKSET_INACTIVE;
         SLM_link_table[i].emergency = FALSE;
         SLM_link_table[i].active_status = LINK_INACTIVE;
         SLM_link_table[i].activation = FALSE;
         SLM_link_table[i].first_failure = FALSE;
         SLM_link_table[i].act_rest_unsuccessful = FALSE;
      }
      else {
         SLM_link_table[i].connected = FALSE;
         SLM_link_table[i].link_slc = 0;
         SLM_link_table[i].stream_num = 0;
         SLM_link_table[i].SP_connected = 0;
         SLM_link_table[i].link_objid = 0;
         SLM_link_table[i].linkset_status = LINKSET_INACTIVE;
         SLM_link_table[i].emergency = FALSE;
         SLM_link_table[i].active_status = LINK_INACTIVE;
         SLM_link_table[i].activation = FALSE;
         SLM_link_table[i].first_failure = FALSE;
         SLM_link_table[i].act_rest_unsuccessful = FALSE;
      }
   }


   /*============================================================ */
   /* create the user_availability table                          */
   /*============================================================ */

   for (i = 0; i < 16; i++) {
      user_availability_table[i].available = TRUE;
      user_availability_table[i].user_part_id = i;
      user_availability_table[i].unavail_cause = 0;
   }

   /* activate all connected links */
   for (i = 0; i < SYS_LINK_LIMIT; i++) {
      if (dv_HMRT_link_table[i].connected) {
         LLSC_LSAC_ACTIVATE_LINK(dv_HMRT_link_table[i].link_slc);
      }
   }
```

APPENDIX F. EXECUTIVES FOR “RECEIVE” STATE

EXECUTIVES FOR “RECEIVE” STATE

```
/* service the interrupt by acquiring the arriving packet */
i = op_intrpt_strm();
pkptr = op_pk_get(i);
op_pk_fd_get(pkptr, FD_INDEX_6TH, &j);

/* packet came from upper layer, L4, try to store message in HMRT input buffer
*/
if ((i >= 0) && (i <= 9)) {
   if (dv_HMRT_state == STATE_HMRT_IDLE) {
      if (op_subq_pk_insert(HMRT_MSU_BUFFER, pkptr, OPC_QPOS_TAIL) !=
OPC_QINS_OK) {
         op_pk_destroy(pkptr);
      }
   }
}
/* packet came from lower layer, L2, try to store message in HMDC input buffer
*/
else if ((i >= 10) && (i <= 31)) {
   if (dv_HMDC_state == STATE_HMDC_IDLE) {
      if (op_subq_pk_insert(HMDC_MSU_BUFFER, pkptr, OPC_QPOS_TAIL) !=
      OPC_QINS_OK) {
         op_pk_destroy(pkptr);
      }
   }
}
else {
   op_pk_destroy(pkptr);
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G. EXECUTIVES FOR "ext_msg" STATE

EXECUTIVES FOR "ext_msg" STATE

```
/* ======================================================================*/
/* Signaling System 7 (SS7), Message Transfer Part Level 3 (MTP3)       */
/* ITU-T Recommendation Q.704, Jul 1996                                 */
/* Note:                                                                */
/*    This state receives all incoming messages for MTP3.               */
/*    Messages here do not include SS7 Signaling Units.                 */
/*    All Messages intended for this level are received here and        */
/*    being recognized as an interrupt with type "OPC_INTRPT_REMOTE".   */
/*    The "code" field of the interrupt are then retrieved to determine */
/*    the message and if necessary, the "ici" field of the interrupt    */
/*    can also be read (if more data are also sent).                    */
/*    All received messages are then queued into message buffer of the  */
/*    respective functional block for processing.                       */
/* ======================================================================*/

iciptr_input = op_intrpt_ici();
op_ici_attr_get(iciptr_input, "data", &ici_data);

switch (op_intrpt_code()) {

   case MSG_LSC_LSAC_IN_SERVICE:
      for (i = 0; i < SYS_LINK_LIMIT; i++) {
         if (SLM_link_table[i].link_objid == ici_data) {
            LSC_LSAC_IN_SERVICE(SLM_link_table[i].link_slc);
         }
      }
      break;

   case MSG_LSC_LSAC_OUT_SERVICE:
      for (i = 0; i < SYS_LINK_LIMIT; i++) {
         if (SLM_link_table[i].link_objid == ici_data) {
            LSC_LSAC_OUT_SERVICE(SLM_link_table[i].link_slc);

         }
      }
      break;

   case MSG_LSC_LSAC_REMOTE_PRO_OUTAGE:
       for (i = 0; i < SYS_LINK_LIMIT; i++) {
         if (SLM_link_table[i].link_objid == ici_data) {
            LSC_LSAC_REMOTE_PRO_OUTAGE(SLM_link_table[i].link_slc);

         }
      }
      break;
case MSG_LSC_LSAC_REMOTE_PRO_RECOVERED:
       for (i = 0; i < SYS_LINK_LIMIT; i++) {
         if (SLM_link_table[i].link_objid == ici_data) {
            LSC_LSAC_REMOTE_PRO_RECOVERED(SLM_link_table[i].link_slc);

         }
      }
      break;
}
```

APPENDIX H. EXECUTIVES FOR "HMDC" STATE

EXECUTIVES FOR "HMDC" STATE

```
/* ===================================================================*/
/* Message Discrimination (HMDC), Signaling Message Handling (SMH)     */
/* Signalling System 7 (SS7), Message Transfer Part Level 2 (MTP3)     */
/* This defines all HMDC processing, which includes handling of incoming*/
/* internal messages and discrimination of MSUs from level 2.          */
/* Messages are proccessed in FIFO order.                              */
/* Messages at the head of the array are considered as earlier.        */
/* Each message is processed according to the algorithm                */
/* as defined in ITU-T Q.704 (Jul 96), Fig 24.                         */
/* ===================================================================*/

switch (op_intrpt_code()) {
   case HMDC_MSU:

      /* Service MSU received from level 2 interrupt           */
      /* If incoming message exists, remove the message packet */
      /* from the buffer and process it                        */
      /* The first field in each packet is always the message  */
      /* code, it can be followed by one or more fields        */
      /* depending on the message                              */

      if (!op_subq_empty(HMDC_MSU_BUFFER)) {
         pkptr = op_subq_pk_remove(HMDC_MSU_BUFFER, OPC_QPOS_HEAD);
         if (dv_HMDC_state == STATE_HMDC_IDLE) {
            op_pk_fd_get(pkptr, FD_INDEX_3RD, &destSP);

            /* insert into HMDT buffer if this is for ownself        */
            /* else send the MSU for routing unless SP is restarting */
            if (destSP == own_PC) {
               if (op_subq_pk_insert(HMDT_MSU_BUFFER, pkptr,
               OPC_QPOS_TAIL) != OPC_QINS_OK) {
                  op_pk_destroy(pkptr);

               }
               dv_HMDC_state = STATE_HMDC_IDLE;
            }
            else {
               if (dv_HMDC_own_SP_restarting_flag) {
                  op_pk_destroy(pkptr);
               }
               else {
                  if (op_subq_pk_insert(HMRT_MSU_BUFFER, pkptr,
                  OPC_QPOS_TAIL) != OPC_QINS_OK) {
                     op_pk_destroy(pkptr);
                  }
               }
            }
         }
      }
```

```
        /* schedule for service of next MSU */
        op_intrpt_schedule_self(op_sim_time() + HMDC_MSU_service_time,
        HMDC_MSU);
        break;

    case HMDC_MSG:

        /* Internal Message Interrupt                                */
        /* If incoming message exists, remove the message packet */
        /* from the buffer and process it                            */
        /* The first field in each packet is always the message  */
        /* code, it can be followed by one or more fields        */
        /* depending on the message                                  */

        if (!op_subq_empty(HMDC_BUFFER)) {
           pkptr = op_subq_pk_remove(HMDC_BUFFER, OPC_QPOS_HEAD);
           op_pk_fd_get(pkptr, FD_INDEX_1ST, &msg_code);

           switch (msg_code) {

               case MSG_TPRC_HMDC_RESTART_BEGINS:
                  if (dv_HMDC_state == STATE_HMDC_IDLE) {
                     dv_HMDC_own_SP_restarting_flag = TRUE;
                  }
                  break;

               case MSG_TPRC_HMDC_RESTART_ENDS:
                  if (dv_HMDC_state == STATE_HMDC_IDLE) {
                     dv_HMDC_own_SP_restarting_flag = FALSE;
                  }
                  break;
           }
           op_pk_destroy(pkptr);
        }
        op_intrpt_schedule_self(op_sim_time() + HMDC_service_time,
        HMDC_MSG);
        break;
}
```

APPENDIX I. EXECUTIVES FOR “HMDT” STATE

EXECUTIVES FOR “HMDT” STATE

```
/* ==================================================================*/
/* Message Distribution (HMDT), Signaling Message Handling (SMH)      */
/* Signalling System 7 (SS7), Message Transfer Part Level 3 (MTP3)    */
/* This defines all HMDT processing which includes handling of incoming */
/* internal messages and distribution of MSUs from level 2.           */
/* Messages are proccessed in FIFO order.                             */
/* Messages at the head of the array are considered as earlier.       */
/* Each message are processed according to the algorithm              */
/* as defined in ITU-T Q.704 (Jul 96), Fig 25.                        */
/* ==================================================================*/

switch (op_intrpt_code()) {
   case HMDT_MSU:

      /* Service MSU received from level 2                          */
      /* If incoming message exists, remove the message packet */
      /* from the buffer and process it                              */
      /* The first field in each packet is always the message  */
      /* code, it can be followed by one or more fields              */
      /* depending on the message                                    */

      if (!op_subq_empty(HMDT_MSU_BUFFER)) {
         pkptr = op_subq_pk_remove(HMDT_MSU_BUFFER, OPC_QPOS_HEAD);
         if (dv_HMDT_state == STATE_HMDT_IDLE) {

            /* read the Service Indicator, SI field */
            op_pk_fd_get(pkptr, FD_INDEX_1ST, &service_indicator);

            switch (service_indicator) {
               case SIO_SI_SIG_NET_MGMT:

                  /* read the HOH1 field */
                  op_pk_fd_get(pkptr, FD_INDEX_6TH, &h0h1);
                  switch (h0h1) {
                     case MSG_SNM_CHANGEOVER_ORDER:
                        SNM_CHANGEOVER_ORDER(pkptr);
                        break;

                     case MSG_SNM_CHANGEOVER_ACK:
                        SNM_CHANGEOVER_ACK(pkptr);
                        break;

                     case MSG_SNM_CHANGEBACK_DECLARATION:
                        SNM_CHANGEBACK_DECLARATION(pkptr);
                        break;

                     case MSG_SNM_CHANGEBACK_ACK:
                        SNM_CHANGEBACK_ACK(pkptr);
                        break;
                     case MSG_SNM_EMERGENCY_CHANGEOVER_ORDER:
```

```
        SNM_EMERGENCY_CHANGEOVER_ORDER(pkptr);
        break;

    case MSG_SNM_EMERGENCY_CHANGEOVER_ACK:
        SNM_EMERGENCY_CHANGEOVER_ACK(pkptr);
        break;

    case MSG_SNM_TRANSFER_PROHIBITED:
        SNM_TRANSFER_PROHIBITED(pkptr);
        break;

    case MSG_SNM_TRANSFER_ALLOWED:
        SNM_TRANSFER_ALLOWED(pkptr);
        break;

    case MSG_SNM_TRANSFER_RESTRICTED:
        SNM_TRANSFER_RESTRICTED(pkptr);
        break;

    case MSG_SNM_SIG_ROUTE_SET_TEST_PROHIBITED:
        SNM_SIG_ROUTE_SET_TEST_PROHIBITED(pkptr);
        break;

    case MSG_SNM_SIG_ROUTE_SET_TEST_RESTRICTED:
        SNM_SIG_ROUTE_SET_TEST_RESTRICTED(pkptr);
        break;

    case MSG_SNM_LINK_INHIBIT:
        SNM_LINK_INHIBIT(pkptr);
        break;

    case MSG_SNM_LINK_UNINHIBIT:
        SNM_LINK_UNINHIBIT(pkptr);
        break;

    case MSG_SNM_LINK_INHIBIT_ACK:
        SNM_LINK_INHIBIT_ACK(pkptr);
        break;

    case MSG_SNM_LINK_UNINHIBIT_ACK:
        SNM_LINK_UNINHIBIT_ACK(pkptr);
        break;

    case MSG_SNM_LINK_INHIBIT_DENIED:
        SNM_LINK_INHIBIT_DENIED(pkptr);
        break;

    case MSG_SNM_LINK_FORCE_UNINHIBIT:
        SNM_LINK_FORCE_UNINHIBIT(pkptr);
        break;

    case MSG_SNM_LINK_LOCAL_INHIBIT_TEST:
        SNM_LINK_LOCAL_INHIBIT_TEST(pkptr);
        break;

    case MSG_SNM_LINK_REMOTE_INHIBIT_TEST:
            SNM_LINK_REMOTE_INHIBIT_TEST(pkptr);
            break;

        case MSG_SNM_TRAFFIC_RESTART_ALLOWED:
            SNM_TRAFFIC_RESTART_ALLOWED(pkptr);
            break;

        case MSG_SNM_DATALINK_CONNECTION_ORDER:
            SNM_DATALINK_CONNECTION_ORDER(pkptr);
            break;

        case MSG_SNM_DATALINK_CONNECTION_SUCCESSFUL:
            SNM_DATALINK_CONNECTION_SUCCESSFUL(pkptr);
            break;

        case MSG_SNM_DATALINK_CONNECTION_NOT_SUCCESSFUL:
            SNM_DATALINK_CONNECTION_NOT_SUCCESSFUL(pkptr);
            break;

        case MSG_SNM_DATALINK_CONNECTION_NOT_POSSIBLE:
            SNM_DATALINK_CONNECTION_NOT_POSSIBLE(pkptr);
            break;

        case MSG_SNM_TRANSFER_CONTROLLED:
            SNM_TRANSFER_CONTROLLED(pkptr);
            break;

        case MSG_SNM_SIG_ROUTE_SET_CONGESTION_TEST:
            SNM_SIG_ROUTE_SET_CONGESTION_TEST(pkptr);
            break;

        case MSG_SNM_USER_PART_UNAVAILABLE:
            SNM_USER_PART_UNAVAILABLE(pkptr);
            break;

        default:
            op_pk_destroy(pkptr);
            break;
    }
    break;

case SIO_SI_SCCP:
    if (user_availability_table[3].available) {
        op_pk_send_forced(pkptr, SCCP_INPUT_STREAM);
    }
    else {
        op_pk_destroy(pkptr);
    }
    break;
```

```
case SIO_SI_TEL_UP:
   if (user_availability_table[4].available) {
      op_pk_send_forced(pkptr, TEL_UP_INPUT_STREAM);
   }
   else {
      op_pk_destroy(pkptr);
   }
   break;

case SIO_SI_ISDN_UP:
   if (user_availability_table[5].available) {
      op_pk_send_forced(pkptr, ISDN_UP_INPUT_STREAM);
   }
   else {
      op_pk_destroy(pkptr);
   }
   break;

case SIO_SI_DATA_UP_CCR:
   if (user_availability_table[6].available) {
      op_pk_send_forced(pkptr, DATA_UP_CCR_INPUT_STREAM);
   }
   else {
      op_pk_destroy(pkptr);
   }
   break;

case SIO_SI_DATA_UP_FRC:
   if (user_availability_table[6].available) {
      op_pk_send_forced(pkptr, DATA_UP_FRC_INPUT_STREAM);
   }
   else {
      op_pk_destroy(pkptr);
   }
   break;

case SIO_SI_BISDN_UP:
   if (user_availability_table[9].available) {
      op_pk_send_forced(pkptr, BISDN_UP_INPUT_STREAM);
   }
   else {
      op_pk_destroy(pkptr);
   }
   break;

case SIO_SI_SAT_ISDN_UP:
   if (user_availability_table[10].available) {
      op_pk_send_forced(pkptr, SAT_ISDN_UP_INPUT_STREAM);
   }
   else {
      op_pk_destroy(pkptr);
   }
   break;
```

```
            default:
               op_pk_destroy(pkptr);
               break;
         }
      }
      else if (dv_HMDT_state == STATE_HMDT_OWN_SP_RESTARTING) {

         /* read the Service Indicator, SI field */
         op_pk_fd_get(pkptr, FD_INDEX_1ST, &service_indicator);

         switch (service_indicator) {
            case SIO_SI_SIG_NET_MGMT:

               /* read the HOH1 field */
               op_pk_fd_get(pkptr, FD_INDEX_6TH, &h0h1);
               switch (h0h1) {
                  case MSG_SNM_TRANSFER_PROHIBITED:
                     SNM_TRANSFER_PROHIBITED(pkptr);
                     break;

                  case MSG_SNM_TRANSFER_ALLOWED:
                     SNM_TRANSFER_ALLOWED(pkptr);
                     break;

                  case MSG_SNM_TRANSFER_RESTRICTED:
                     SNM_TRANSFER_RESTRICTED(pkptr);
                     break;

                  case MSG_SNM_TRAFFIC_RESTART_ALLOWED:
                     SNM_TRAFFIC_RESTART_ALLOWED(pkptr);
                     break;

                  default:
                     op_pk_destroy(pkptr);
                     break;
               }
               break;

            case SIO_SI_NET_TEST_MAINT:

               /* not implemented */
               op_pk_destroy(pkptr);
               break;

            default:
               op_pk_destroy(pkptr);
               break;
         }
      }
   }

   /* schedule for service of next MSU */
   op_intrpt_schedule_self(op_sim_time() + HMDT_MSU_service_time,
   HMDT_MSU);
   break;
```

```
    case HMDT_MSG:

        /* Internal Message Interrupt                            */
        /* If incoming message exists, remove the message packet */
        /* from the buffer and process it                        */
        /* The first field in each packet is always the message  */
        /* code, it can be followed by one or more fields        */
        /* depending on the message                              */


        if (!op_subq_empty(HMDT_BUFFER)) {
           pkptr = op_subq_pk_remove(HMDT_BUFFER, OPC_QPOS_HEAD);
           op_pk_fd_get(pkptr, FD_INDEX_1ST, &msg_code);

           switch (msg_code) {

               case MSG_TPRC_HMDT_RESTART_BEGINS:
                  if (dv_HMDT_state == STATE_HMDT_IDLE) {
                     dv_HMDT_state = STATE_HMDT_OWN_SP_RESTARTING;
                  }
                  break;

               case MSG_TPRC_HMDT_RESTART_ENDS:
                  if (dv_HMDT_state == STATE_HMDC_OWN_SP_RESTARTING) {
                     dv_HMDT_state = STATE_HMDT_IDLE;
                  }
                  break;
           }
           op_pk_destroy(pkptr);
        }
        op_intrpt_schedule_self(op_sim_time() + HMDT_service_time,
        HMDT_MSG);
        break;
}
```

APPENDIX J. EXECUTIVES FOR "HMRT" STATE

EXECUTIVES FOR "HMRT" STATE

```
/* ==================================================================*/
/* Message Routing (HMRT), Signaling Message Handling (SMH)         */
/* Signalling System 7 (SS7), Message Transfer Part Level 3 (MTP3)   */
/* This defines all HMRT processing which includes handling of incoming */
/* internal messages and routing of MSUs from level 4 and 2.        */
/* Messages are proccessed in FIFO order.                            */
/* Messages at the head of the array are considered as earlier.      */
/* Each message are processed according to the algorithm             */
/* as defined in ITU-T Q.704 (Jul 96), Fig 26.                       */
/* ==================================================================*/


switch (op_intrpt_code()) {
   case HMRT_MSU:

      /* Service MSU received                                 */
      /* If incoming message exists, remove the message packet */
      /* from the buffer and process it                       */
      /* The first field in each packet is always the message  */
      /* code, it can be followed by one or more fields       */
      /* depending on the message                             */

      if (!op_subq_empty(HMRT_MSU_BUFFER)) {
         pkptr = op_subq_pk_remove(HMRT_MSU_BUFFER, OPC_QPOS_HEAD);

         /* routing label fields */
         op_pk_fd_get(pkptr, FD_INDEX_1ST, &service_indicator);
         op_pk_fd_get(pkptr, FD_INDEX_3RD, &destSP);
         op_pk_fd_get(pkptr, FD_INDEX_4TH, &origSP);
         op_pk_fd_get(pkptr, FD_INDEX_5TH, &sig_link_selection);

         /* MSUs generated by own SP */
         if (origSP == own_PC) {

            /* MSUs generated by own SP's Network Management */
            if (service_indicator == SIO_SI_SIG_NET_MGMT) {
               op_pk_fd_get(pkptr, FD_INDEX_6TH, &h0h1);

               if (dv_HMRT_state = STATE_HMRT_IDLE) {

                  /* MSU from STM */
                  if (((h0h1 >= 0x10) && (h0h1 <= 0x1F)) || ((h0h1 >=
                  0x20) && (h0h1 <= 0x2F)) || ((h0h1 >= 0x60) && (h0h1
                  <= 0x6F)) || ((h0h1 >= 0x70) && (h0h1 <= 0x7F))) {
                     if (sf_rt_HMRT_is_DPC_adj_and_restarting(destSP)) {
                        if (h0h1 == 0x71) {
                           sls_assign_ptr =
                           sf_rt_HMRT_pick_a_link_in_linkset_for_tx(destSP);
                           op_pk_send(pkptr,
                           dv_HMRT_link_table[sls_assign_ptr->
            link_slc].stream_num);
         }
         else {
            discard_MSU(pkptr);
         }
      }
   }

   /* MSU from SRM */
   else if (((h0h1 >= 0x30) && (h0h1 <= 0x3F)) || ((h0h1 >=
   0x40) && (h0h1 <= 0x4F)) ||((h0h1 >= 0x50) && (h0h1 <=
   0x5F)) || ((h0h1 >= 0xA0) && (h0h1 <= 0xAF))) {
      if (sf_rt_HMRT_is_DPC_adj_and_restarting(destSP)) {
         if (sf_rt_HMRT_is_route_available(destSP)) {
            if ((h0h1 == 0x41) || (h0h1 == 0x43) || (h0h1 ==
            0x45)) {
               if (sf_rt_HMRT_is_DPC_congested(destSP)) {
                  HMRT_TSFC_MSG_FOR_CONGESTED_DEST(origSP,
                  destSP);
               }
               sls_assign_ptr =
               sf_rt_HMRT_pick_any_link_for_tx();
               op_pk_send(pkptr,
               dv_HMRT_link_table[sls_assign_ptr->
               link_slc].stream_num);
            }
            else {
               discard_MSU(pkptr);
            }
         }
         else {
            HMRT_MGMT_MSG_RECEIVED_FOR_
            INACCESSIBLE_SP(origSP, destSP);
            HMRT_RTPC_MSG_RECEIVED_FOR_
            INACCESSIBLE_SP(origSP, destSP);
            discard_MSU(pkptr);
         }
      }
      else {
         if (sf_rt_HMRT_is_DPC_congested(destSP)) {
            HMRT_TSFC_MSG_FOR_CONGESTED_DEST(origSP, destSP);
         }
         sls_assign_ptr = sf_rt_HMRT_pick_any_link_for_tx();
         op_pk_send(pkptr, dv_HMRT_link_table[sls_assign_ptr
         ->link_slc].stream_num);
      }
   }

   /* MSU from SLM */
   else if ((h0h1 >= 0x80) && (h0h1 <= 0x8F)) {
      if (sf_rt_HMRT_is_route_available(destSP)) {
         if (sf_rt_HMRT_is_DPC_adj_and_restarting(destSP)) {
            discard_MSU(pkptr);
         }
         else {
            sls_assign_ptr =
                sf_rt_HMRT_pick_any_link_for_tx();
                op_pk_send(pkptr,
                dv_HMRT_link_table[sls_assign_ptr->
                link_slc].stream_num);
            }
        }
        else {
            HMRT_MGMT_MSG_RECEIVED_FOR_INACCESSIBLE_SP(origSP,
            destSP);
            HMRT_RTPC_MSG_RECEIVED_FOR_INACCESSIBLE_SP(origSP,
            destSP);
            discard_MSU(pkptr);
        }
    }
}
else if (dv_HMRT_state = STATE_HMRT_OWN_SP_RESTARTING) {

    /* MSU from STM */
    if (((h0h1 >= 0x10) && (h0h1 <= 0x1F)) || ((h0h1 >= 0x20)
    && (h0h1 <= 0x2F)) || ((h0h1 >= 0x60) && (h0h1 <= 0x6F))
    || ((h0h1 >= 0x70) && (h0h1 <= 0x7F))) {
        if (h0h1 == 0x71) {
            if (sf_rt_HMRT_is_route_available(destSP)) {
                if (sf_rt_HMRT_is_DPC_congested(destSP)) {
                    HMRT_TSFC_MSG_FOR_CONGESTED_DEST(origSP,
                    destSP);
                }
                sls_assign_ptr =
              sf_rt_HMRT_pick_a_direct_link_for_
              tx(sig_link_selection, destSP);
                op_pk_send(pkptr,
                dv_HMRT_link_table[sls_assign_ptr->
                link_slc].stream_num);
            }
            else {
                discard_MSU(pkptr);
            }
        }
        else {
            discard_MSU(pkptr);
        }
    }

    /* MSU from SRM */
    else if (((h0h1 >= 0x30) && (h0h1 <= 0x3F)) || ((h0h1 >=
    0x40) && (h0h1 <= 0x4F)) ||((h0h1 >= 0x50) && (h0h1 <=
    0x5F)) || ((h0h1 >= 0xA0) && (h0h1 <= 0xAF))) {
        if ((h0h1 == 0x41) || (h0h1 == 0x43) || (h0h1 == 0x45)){
            if (sf_rt_HMRT_is_route_available(destSP)) {
                if (sf_rt_HMRT_is_DPC_congested(destSP)) {
                    HMRT_TSFC_MSG_FOR_CONGESTED_DEST(origSP,
                    destSP);
                }
                sls_assign_ptr = sf_rt_HMRT_pick_a_direct_link_for_
                tx(sig_link_selection, destSP);
                op_pk_send(pkptr, dv_HMRT_link_table[sls_assign_ptr
               ->link_slc].stream_num);
            }
            else {
               discard_MSU(pkptr);
            }
         }
         else {
            discard_MSU(pkptr);
         }
      }

      /* MSU from SLM */
      else if ((h0h1 >= 0x80) && (h0h1 <= 0x8F)) {
         discard_MSU(pkptr);
      }
   }
}

/* MSUs generated by own SP's Network Testing and Maintenance */
else if (service_indicator == SIO_SI_NET_TEST_MAINT) {
   op_pk_fd_get(pkptr, FD_INDEX_6TH, &h0h1);


   /* MSUs from SLTC */
   if (h0h1 == CODE_H0H1_SNTM_SIG_LINK_TEST_CONTROL_MSG) {
      if ((dv_HMRT_state == STATE_HMRT_IDLE) ||
          (dv_HMRT_state == STATE_HMRT_OWN_SP_RESTARTING)) {
         op_pk_send(pkptr,
         dv_HMRT_link_table[sig_link_selection].stream_num);
      }
   }
}

/* MSUs generated by own SP's Level 4 MTP Testing User Part */
else if (service_indicator == SIO_SI_MTP_TESTING_UP) {

   if (dv_HMRT_state == STATE_HMRT_IDLE) {

      /* details on routing of this type of MSU not available */
      /* so they are discarded by default                     */
      discard_MSU(pkptr);
   }
   else if (dv_HMRT_state == STATE_HMRT_OWN_SP_RESTARTING) {
      discard_MSU(pkptr);
   }
}

/* MSUs generated by own SP's other Level 4 users */
else {
   if (dv_HMRT_state == STATE_HMRT_IDLE) {
      if (sf_rt_HMRT_is_route_available(destSP)) {
         if (sf_rt_HMRT_is_DPC_adj_and_restarting(destSP)) {
            discard_MSU(pkptr);
         }
         else {
            if (sf_rt_HMRT_is_DPC_congested(destSP)) {
                    HMRT_TSFC_MSG_FOR_CONGESTED_DEST(origSP, destSP);
                }
                sls_assign_ptr = sf_rt_HMRT_pick_a_link_for_
                tx(destSP, sig_link_selection);
                op_pk_send(pkptr, dv_HMRT_link_table[sls_assign_ptr->
                link_slc].stream_num);
            }
        }
        else {
            HMRT_MGMT_MSG_RECEIVED_FOR_INACCESSIBLE_SP
            (origSP, destSP);
            HMRT_RTPC_MSG_RECEIVED_FOR_INACCESSIBLE_SP
            (origSP, destSP);
            discard_MSU(pkptr);
        }
    }
    else if (dv_HMRT_state == STATE_HMRT_OWN_SP_RESTARTING) {
        discard_MSU(pkptr);
    }
}
}

/* external MSUs for routing from HMDC */
else {
    if (dv_HMRT_state == STATE_HMRT_IDLE) {
        if (sf_rt_HMRT_is_route_available(destSP)) {
            if (sf_rt_HMRT_is_DPC_adj_and_restarting(destSP)) {
                discard_MSU(pkptr);
            }
            else {
                if (sf_rt_HMRT_is_DPC_congested(destSP)) {
                    HMRT_TSFC_MSG_FOR_CONGESTED_DEST(origSP, destSP);
                }
                sls_assign_ptr = sf_rt_HMRT_pick_a_link_for_tx(destSP,
                sig_link_selection);
                op_pk_send(pkptr, dv_HMRT_link_table[sls_assign_ptr->
                link_slc].stream_num);
            }
        }
        else {
            HMRT_MGMT_MSG_RECEIVED_FOR_INACCESSIBLE_SP(origSP, destSP);
            HMRT_RTPC_MSG_RECEIVED_FOR_INACCESSIBLE_SP(origSP, destSP);
            discard_MSU(pkptr);
        }
    }
    else if (dv_HMRT_state == STATE_HMRT_OWN_SP_RESTARTING) {
        discard_MSU(pkptr);
    }
}
}

/* schedule for service of next MSU */
op_intrpt_schedule_self(op_sim_time() + HMRT_MSU_service_time,
HMRT_MSU);
break;
```

```
case HMRT_MSG:

   /* Internal Message Interrupt                              */
   /* If incoming message exists, remove the message packet */
   /* from the buffer and process it                        */
   /* The first field in each packet is always the message  */
   /* code, it can be followed by one or more fields        */
   /* depending on the message                              */

   if (!op_subq_empty(HMRT_BUFFER)) {
      pkptr = op_subq_pk_remove(HMRT_BUFFER, OPC_QPOS_HEAD);
      op_pk_fd_get(pkptr, FD_INDEX_1ST, &msg_code);

      switch (msg_code) {

          case MSG_TSRC_HMDT_ADJ_SP_RESTART:
             if (dv_HMRT_state == STATE_HMRT_IDLE) {
                op_pk_fd_get(pkptr, FD_INDEX_2ND, &adjSP);
                /* mark the adjSP restarting */
                for (i = 0; i < link_total; i++) {
                   if (dv_HMRT_link_table[i].SP_connected == adjSP) {
                      dv_HMRT_link_table[i].adjSP_restarting = TRUE;
                   }
                }
             }
             break;

          case MSG_TPRC_HMRT_RESTART_BEGINS:
             if (dv_HMRT_state == STATE_HMRT_IDLE) {
                dv_HMRT_state = STATE_HMRT_OWN_SP_RESTARTING;
             }
             break;

          case MSG_TPRC_HMRT_RESTART_ENDS:
             if (dv_HMRT_state == STATE_HMRT_OWN_SP_RESTARTING) {
                /* clear all adjacent SP restart mark */
                for (i = 0; i < SYS_LINK_LIMIT; i++) {
                   dv_HMRT_link_table[i].adjSP_restarting = FALSE;
                }
                dv_HMRT_state = STATE_HMRT_IDLE;
             }
             break;

          case MSG_TCRC_HMRT_UPDATE_ROUTING_TABLES:
             if (dv_HMRT_state == STATE_HMRT_IDLE) {
                sf_rt_HMRT_update_routing_tables();
             }
             break;

          case MSG_TFRC_HMRT_UPDATE_ROUTING_TABLES:
             if (dv_HMRT_state == STATE_HMRT_IDLE) {
                sf_rt_HMRT_update_routing_tables();
             }
             break;
      }
```

```
        op_pk_destroy(pkptr);
          }
          op_intrpt_schedule_self(op_sim_time() + HMRT_service_time, HMRT_MSG);
          break;
    }
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX K. EXECUTIVES FOR "LLSC" STATE

EXECUTIVES FOR "LLSC" STATE

```
/* ======================================================================*/
/* Link Set Control (LLSC), Signaling Link Management(SLM)              */
/* Signalling System 7 (SS7), Message Transfer Part Level 3 (MTP3)      */
/* Each message are processed according to the algorithm                */
/* as defined in ITU-T Q.704 (Jul 96), Fig 36.                           */
/* ======================================================================*/

if (!op_subq_empty(LLSC_BUFFER)) {
   pkptr = op_subq_pk_remove(LLSC_BUFFER, OPC_QPOS_HEAD);
   op_pk_fd_get(pkptr, FD_INDEX_1ST, &msg_code);

   switch (msg_code) {

       case MSG_MGMT_LLSC_ACTIVATE_LINK_SET:

          /* activate all inactive links in the link set */
          op_pk_fd_get(pkptr, FD_INDEX_2ND, &adjSP);
          if (sf_rt_SLM_linkset_status(adjSP) == LINKSET_INACTIVE) {
             for (i = 0; i < SYS_LINK_LIMIT; i++) {
                if ((SLM_link_table[i].connected) &&
                    (SLM_link_table[i].SP_connected == adjSP) &&
                    (SLM_link_table[i].active_status == LINK_INACTIVE)) {
                       LLSC_LSAC_ACTIVATE_LINK(SLM_link_table[i].link_slc);
                }
             }
             sf_rt_SLM_set_linkset_active(adjSP);
          }
          else if (sf_rt_SLM_linkset_status(adjSP) == LINKSET_ACTIVE) {
             for (i = 0; i < SYS_LINK_LIMIT; i++) {
                if ((SLM_link_table[i].connected) &&
                    (SLM_link_table[i].SP_connected == adjSP) &&
                    (SLM_link_table[i].active_status == LINK_INACTIVE)) {
                       LLSC_LSAC_ACTIVATE_LINK(SLM_link_table[i].link_slc);
                }
             }
          }
          break;

       case MSG_TPRC_LLSC_RESTART_BEGINS:

          /* activate all inactive links in the link set */
          op_pk_fd_get(pkptr, FD_INDEX_2ND, &adjSP);
          if (sf_rt_SLM_linkset_status(adjSP) == LINKSET_INACTIVE) {
             for (i = 0; i < SYS_LINK_LIMIT; i++) {
                if ((SLM_link_table[i].connected) &&
                    (SLM_link_table[i].SP_connected == adjSP) &&
                    (SLM_link_table[i].active_status == LINK_INACTIVE)) {
                       LLSC_LSAC_ACTIVATE_LINK(SLM_link_table[i].link_slc);
                }
             }
```

```
        sf_rt_SLM_set_linkset_active(adjSP);
    }
    else if (sf_rt_SLM_linkset_status(adjSP) == LINKSET_ACTIVE) {
        for (i = 0; i < SYS_LINK_LIMIT; i++) {
            if ((SLM_link_table[i].connected) &&
                 (SLM_link_table[i].SP_connected == adjSP) &&
                 (SLM_link_table[i].active_status == LINK_INACTIVE)) {
                    LLSC_LSAC_ACTIVATE_LINK(SLM_link_table[i].link_slc);
            }
        }
    }
    break;

case MSG_TPRC_LLSC_RESTART_ENDS:

    /* inform every link in the link set */
    op_pk_fd_get(pkptr, FD_INDEX_2ND, &adjSP);
    if (sf_rt_SLM_linkset_status(adjSP) == LINKSET_INACTIVE) {
        for (i = 0; i < SYS_LINK_LIMIT; i++) {
            if ((SLM_link_table[i].connected) &&
                 (SLM_link_table[i].SP_connected == adjSP)) {
                    LLSC_LSAC_RESTART_ENDS(SLM_link_table[i].link_slc);
            }
        }
    }
    break;

case MSG_MGMT_LLSC_DEACTIVATE_LINK_SET:

    /* deactivate all links in the link set */
    op_pk_fd_get(pkptr, FD_INDEX_2ND, &adjSP);
    if (sf_rt_SLM_linkset_status(adjSP) == LINKSET_ACTIVE) {
        for (i = 0; i < SYS_LINK_LIMIT; i++) {
            if ((SLM_link_table[i].connected) &&
                 (SLM_link_table[i].SP_connected == adjSP) &&
                 (SLM_link_table[i].active_status != LINK_INACTIVE)) {
                    LLSC_LSAC_DEACTIVATE_LINK(SLM_link_table[i].link_slc);
            }
        }
        sf_rt_SLM_cancel_linkset_active(adjSP);
    }
    break;

case MSG_LSAC_LLSC_LINK_ACTIVE:
    op_pk_fd_get(pkptr, FD_INDEX_2ND, &adjSP);
    /* do nothing since all links are expected to be activated */
    break;

case MSG_LSAC_LLSC_LINK_INACTIVE:
  op_pk_fd_get(pkptr, FD_INDEX_2ND, &adjSP);
    /* do nothing since all links are expected to be activated */
    break;
```

```
case MSG_LSAC_LLSC_LINK_FAILED:

    /* try to activate one other links in the the linkset */
    op_pk_fd_get(pkptr, FD_INDEX_2ND, &adjSP);
    if (sf_rt_SLM_linkset_status(adjSP) == LINKSET_ACTIVE) {
        for (i = 0; i < SYS_LINK_LIMIT; i++) {
            if ((SLM_link_table[i].connected) &&
                 (SLM_link_table[i].SP_connected == adjSP) &&
                 (SLM_link_table[i].active_status == LINK_INACTIVE) &&
                 (SLM_link_table[i].link_slc != link_num)) {
                    LLSC_LSAC_ACTIVATE_LINK(SLM_link_table[i].link_slc);
            }
        }
    }
    break;

case MSG_LSAC_LLSC_ACTIVATE_ANOTHER_LINK:

    /* try to activate one other links in the the linkset */
    op_pk_fd_get(pkptr, FD_INDEX_2ND, &adjSP);
    if (sf_rt_SLM_linkset_status(adjSP) == LINKSET_ACTIVE) {
        for (i = 0; i < SYS_LINK_LIMIT; i++) {
            if ((SLM_link_table[i].connected) &&
                 (SLM_link_table[i].SP_connected == adjSP) &&
                 (SLM_link_table[i].active_status == LINK_INACTIVE) &&
                 (SLM_link_table[i].link_slc != link_num)) {
                    LLSC_LSAC_ACTIVATE_LINK(SLM_link_table[i].link_slc);
            }
        }
    }
    break;

case MSG_TSRC_LLSC_EMERGENCY:
    op_pk_fd_get(pkptr, FD_INDEX_2ND, &adjSP);
    if (sf_rt_SLM_linkset_status(adjSP) == LINKSET_ACTIVE) {
        for (i = 0; i < SYS_LINK_LIMIT; i++) {
            if ((SLM_link_table[i].connected) &&
                 (SLM_link_table[i].SP_connected == adjSP)) {
                    LLSC_LSAC_EMERGENCY(SLM_link_table[i].link_slc);
            }
        }
    }
    break;

case MSG_TSRC_LLSC_EMERGENCY_CEASES:
    op_pk_fd_get(pkptr, FD_INDEX_2ND, &adjSP);
    if (sf_rt_SLM_linkset_status(adjSP) == LINKSET_ACTIVE) {
        for (i = 0; i < SYS_LINK_LIMIT; i++) {
            if ((SLM_link_table[i].connected) &&
                 (SLM_link_table[i].SP_connected == adjSP)) {
                    LLSC_LSAC_EMERGENCY_CEASES(SLM_link_table[i].link_slc);
            }
        }
    }
    break;
```

```
        }

        /* destroy the packet */
        op_pk_destroy(pkptr);
    }

    /* schedule interrupt to check and process next LSLA msg */
    op_intrpt_schedule_self(op_sim_time() + LLSC_service_time, LLSC_MSG);
```

APPENDIX L. EXECUTIVES FOR "LSAC" STATE

EXECUTIVES FOR "LSAC" STATE

```
/* ==================================================================*/
/* Signaling Link Activity Control(LSAC), Signaling Link Management(SLM)*/
/* Signalling System 7 (SS7), Message Transfer Part Level 3 (MTP3)      */
/* Each message are processed according to the algorithm               */
/* as defined in ITU-T Q.704 (Jul 96), Fig 37.                          */
/* ==================================================================*/

switch (op_intrpt_code()) {

   case LSAC_MSG:

      if (!op_subq_empty(LSAC_BUFFER)) {
         pkptr = op_subq_pk_remove(LSAC_BUFFER, OPC_QPOS_HEAD);
         op_pk_fd_get(pkptr, FD_INDEX_1ST, &msg_code);
         op_pk_fd_get(pkptr, FD_INDEX_2ND, &link_num);
         switch (msg_code) {

            case MSG_LLSC_LSAC_ACTIVATE_LINK:
               if (SLM_link_table[link_num].active_status == LINK_INACTIVE) {
                  LSAC_LSLA_START_ACTIVATION(link_num);
                  SLM_link_table[link_num].active_status =
                  LINK_ACTIVATING_RESTORING;
                  op_prg_log_entry_write(log_hdle_link_startup, "LSAC:
                  Activating link num:%d, objid:%d", link_num,
                  SLM_link_table[link_num].link_objid);
               }
               break;

            case MSG_MGMT_LSAC_ACTIVATE_LINK:
               if (SLM_link_table[link_num].active_status == LINK_INACTIVE) {
                  LSAC_LSLA_START_ACTIVATION(link_num);
                  SLM_link_table[link_num].active_status =
                  LINK_ACTIVATING_RESTORING;
               }
               break;

            case MSG_LLSC_LSAC_EMERGENCY:
               if (SLM_link_table[link_num].active_status == LINK_INACTIVE) {
                  SLM_link_table[link_num].emergency = TRUE;
               }
               else if (SLM_link_table[link_num].active_status ==
               LINK_ACTIVATING_RESTORING) {
                  SLM_link_table[link_num].emergency = TRUE;
                  mtp3_send_external_msg(SLM_link_table[link_num].link_objid,
                  MSG_LSAC_LSC_EMERGENCY, 0, iciptr_external);
               }
               else if (SLM_link_table[link_num].active_status == LINK_ACTIVE) {
                  SLM_link_table[link_num].emergency = TRUE;
               }
               break;
```

```
case MSG_LLSC_LSAC_EMERGENCY_CEASES:
   if (SLM_link_table[link_num].active_status == LINK_INACTIVE) {
      SLM_link_table[link_num].emergency = FALSE;
   }
   else if (SLM_link_table[link_num].active_status ==
   LINK_ACTIVATING_RESTORING) {
      SLM_link_table[link_num].emergency = FALSE;
      mtp3_send_external_msg(SLM_link_table[link_num].link_objid,
      MSG_LSAC_LSC_EMERGENCY_CEASES, 0, iciptr_external);
   }
   else if (SLM_link_table[link_num].active_status == LINK_ACTIVE) {
      SLM_link_table[link_num].emergency = FALSE;
   }
   break;

case MSG_SLTC_LSAC_SLT_SUCCESSFUL:
   if (SLM_link_table[link_num].active_status ==
   LINK_ACTIVATING_RESTORING) {
      LSAC_TLAC_LINK_IN_SERVICE(link_num);
      LSAC_LLSC_LINK_ACTIVE(link_num);
      SLM_link_table[link_num].first_failure = FALSE;
      SLM_link_table[link_num].act_rest_unsuccessful = FALSE;
      SLM_link_table[link_num].active_status = LINK_ACTIVE;
      op_prg_log_entry_write(log_hdle_link_startup, "LSAC: link in
      service, num: %d", link_num);
   }
   break;

case MSG_LSC_LSAC_IN_SERVICE:
   if (SLM_link_table[link_num].active_status ==
   LINK_ACTIVATING_RESTORING) {
      LSAC_SLTC_START_SLT(link_num);
   }
   break;

case MSG_LSC_LSAC_OUT_SERVICE:
   if (SLM_link_table[link_num].active_status ==
   LINK_ACTIVATING_RESTORING) {
      if (SLM_link_table[link_num].first_failure) {
         SLM_link_table[link_num].first_failure = FALSE;
      }
      else {
         SLM_link_table[link_num].first_failure = TRUE;
      }
      if (SLM_link_table[link_num].activation) {
         SLM_link_table[link_num].activation = FALSE;
         LSAC_LSLA_RESTART_ACTIVATION(link_num);
      }
      else {
         LSAC_LSLR_RESTART_RESTORATION(link_num);
      }
   }
   else if (SLM_link_table[link_num].active_status == LINK_ACTIVE) {
      LSAC_TLAC_LINK_FAILED(link_num);
      mtp3_send_external_msg(SLM_link_table[link_num].link_objid,
            MSG_LSAC_LSC_STOP, 0, iciptr_external);
            LSAC_LLSC_LINK_FAILED(link_num);
            LSAC_LSLR_START_RESTORATION(link_num);
            SLM_link_table[link_num].active_status =
            LINK_ACTIVATING_RESTORING;
        }
        break;

    case MSG_LSLA_LSAC_START_SIG_LINK:
        if (SLM_link_table[link_num].active_status ==
        LINK_ACTIVATING_RESTORING) {
            dv_timer17[last_timer17].data = link_num;
            mtp3_timer_start(&dv_timer17[last_timer17]);
            last_timer17 = mtp3_increment(last_timer17, 1,
            SYS_LINK_LIMIT);
            SLM_link_table[link_num].active_status =
            LINK_ACTIVATING_RESTORING_WAIT;
        }
        break;

    case MSG_LSLR_LSAC_START_SIG_LINK:
        if (SLM_link_table[link_num].active_status ==
        LINK_ACTIVATING_RESTORING) {
            dv_timer17[last_timer17].data = link_num;
            mtp3_timer_start(&dv_timer17[last_timer17]);
            last_timer17 = mtp3_increment(last_timer17, 1,
            SYS_LINK_LIMIT);
            SLM_link_table[link_num].active_status =
            LINK_ACTIVATING_RESTORING_WAIT;
        }
        break;

    case MSG_LSLA_LSAC_ACTIVATION_UNSUCCESSFUL:
        if (SLM_link_table[link_num].active_status ==
        LINK_ACTIVATING_RESTORING) {
            if (!SLM_link_table[link_num].act_rest_unsuccessful) {
                SLM_link_table[link_num].act_rest_unsuccessful = TRUE;
                LSAC_LLSC_ACTIVATE_ANOTHER_LINK(link_num);
            }
        }
        break;

    case MSG_LSLR_LSAC_RESTORATION_UNSUCCESSFUL:
        if (SLM_link_table[link_num].active_status ==
        LINK_ACTIVATING_RESTORING) {
            if (!SLM_link_table[link_num].act_rest_unsuccessful) {
                SLM_link_table[link_num].act_rest_unsuccessful = TRUE;
                LSAC_LLSC_ACTIVATE_ANOTHER_LINK(link_num);
            }
        }
        break;

    case MSG_SLTC_LSAC_SLT_FAILED:
        if (SLM_link_table[link_num].active_status ==
        LINK_ACTIVATING_RESTORING) {
            mtp3_send_external_msg(SLM_link_table[link_num].link_objid,
            MSG_LSAC_LSC_STOP, 0, iciptr_external);
            LSAC_LSLD_DEACTIVATE_LINK(link_num);
            LSAC_LSLA_DEACTIVATE_LINK(link_num);
            LSAC_LLSC_LINK_INACTIVE(link_num);
            if (SLM_link_table[link_num].act_rest_unsuccessful) {
               SLM_link_table[link_num].act_rest_unsuccessful = FALSE;
            }
            else {
               LSAC_LLSC_ACTIVATE_ANOTHER_LINK(link_num);
            }
            SLM_link_table[link_num].active_status = LINK_INACTIVE;
         }
         break;

      case MSG_LLSC_LSAC_DEACTIVATE_LINK:
         if (SLM_link_table[link_num].active_status ==
         LINK_ACTIVATING_RESTORING) {
            mtp3_send_external_msg(SLM_link_table[link_num].link_objid,
            MSG_LSAC_LSC_STOP, 0, iciptr_external);
            LSAC_LSLD_DEACTIVATE_LINK(link_num);
            LSAC_LSLA_DEACTIVATE_LINK(link_num);
            LSAC_LLSC_LINK_INACTIVE(link_num);
            if (SLM_link_table[link_num].act_rest_unsuccessful) {
               SLM_link_table[link_num].act_rest_unsuccessful = FALSE;
            }
            else {
               LSAC_LLSC_ACTIVATE_ANOTHER_LINK(link_num);
            }
            SLM_link_table[link_num].active_status = LINK_INACTIVE;
         }
         else if (SLM_link_table[link_num].active_status == LINK_ACTIVE) {
            LSAC_TLAC_LINK_FAILED(link_num);
            SLM_link_table[link_num].active_status = LINK_ACTIVE_WAIT;
         }
         break;

      case MSG_MGMT_LSAC_DEACTIVATE_LINK:
         if (SLM_link_table[link_num].active_status ==
         LINK_ACTIVATING_RESTORING) {
            mtp3_send_external_msg(SLM_link_table[link_num].link_objid,
            MSG_LSAC_LSC_STOP, 0, iciptr_external);
            LSAC_LSLD_DEACTIVATE_LINK(link_num);
            LSAC_LSLA_DEACTIVATE_LINK(link_num);
            LSAC_LLSC_LINK_INACTIVE(link_num);
            if (SLM_link_table[link_num].act_rest_unsuccessful) {
               SLM_link_table[link_num].act_rest_unsuccessful = FALSE;
            }
            else {
               LSAC_LLSC_ACTIVATE_ANOTHER_LINK(link_num);
            }
            SLM_link_table[link_num].active_status = LINK_INACTIVE;
         }
         else if (SLM_link_table[link_num].active_status == LINK_ACTIVE) {
            LSAC_LLSC_ACTIVATE_ANOTHER_LINK(link_num);
            LSAC_TLAC_LINK_FAILED(link_num);
            SLM_link_table[link_num].active_status = LINK_ACTIVE_WAIT;
```

```
    }
    break;

case MSG_LLSC_LSAC_RESTART_ENDS:
    if (SLM_link_table[link_num].active_status ==
    LINK_ACTIVATING_RESTORING) {
        LSAC_TLAC_LINK_FAILED(link_num);
    }
    break;

case MSG_LSC_LSAC_REMOTE_PRO_RECOVERED:
    if (SLM_link_table[link_num].active_status ==
    LINK_ACTIVATING_RESTORING) {
        LSAC_SLTC_START_SLT(link_num);
    }
    else if (SLM_link_table[link_num].active_status == LINK_ACTIVE) {
        LSAC_TLAC_REMOTE_PRO_RECOVERED(link_num);
    }
    break;

case MSG_LSC_LSAC_REMOTE_PRO_OUTAGE:
    if (SLM_link_table[link_num].active_status == LINK_ACTIVE) {
        LSAC_TLAC_REMOTE_PRO_OUTAGE(link_num);
    }
    break;

case MSG_TCOC_LSAC_STOP_L2:
    if (SLM_link_table[link_num].active_status == LINK_ACTIVE) {
        mtp3_send_external_msg(SLM_link_table[link_num].link_objid,
        MSG_LSAC_LSC_STOP, 0, iciptr_external);
        LSAC_LLSC_LINK_FAILED(link_num);
        LSAC_LSLR_START_RESTORATION(link_num);
        SLM_link_table[link_num].active_status =
        LINK_ACTIVATING_RESTORING;
    }
    break;

case MSG_TLAC_LSAC_CHANGEOVER_ORDER_RECEIVED:
    if (SLM_link_table[link_num].active_status == LINK_ACTIVE) {
        mtp3_send_external_msg(SLM_link_table[link_num].link_objid,
        MSG_LSAC_LSC_STOP, 0, iciptr_external);
        LSAC_LLSC_LINK_FAILED(link_num);
        LSAC_LSLR_START_RESTORATION(link_num);
        SLM_link_table[link_num].active_status =
        LINK_ACTIVATING_RESTORING;
    }
    break;

case MSG_TCOC_LSAC_STM_READY:
    if (SLM_link_table[link_num].active_status == LINK_ACTIVE_WAIT) {
        LSAC_LLSC_LINK_FAILED(link_num);
        mtp3_send_external_msg(SLM_link_table[link_num].link_objid,
        MSG_LSAC_LSC_STOP, 0, iciptr_external);
        LSAC_LSLD_DEACTIVATE_LINK(link_num);
        LSAC_LLSC_LINK_INACTIVE(link_num);
        SLM_link_table[link_num].active_status = LINK_INACTIVE;
```

```
                }
                break;

            case MSG_TLAC_LSAC_FLUSH_BUFFERS:
                if (SLM_link_table[link_num].active_status == LINK_ACTIVE) {
                    mtp3_send_external_msg(SLM_link_table[link_num].link_objid,
                    MSG_LSAC_LSC_FLUSH_BUFFERS, 0, iciptr_external);
                }
                break;

            case MSG_TLAC_LSAC_CONTINUE:
                if (SLM_link_table[link_num].active_status == LINK_ACTIVE) {
                    mtp3_send_external_msg(SLM_link_table[link_num].link_objid,
                    MSG_LSAC_LSC_CONTINUE, 0, iciptr_external);
                }
                break;
        }

        /* destroy the message packet */
        op_pk_destroy(pkptr);
    }

    /* schedule interrupt to check and process next LSAC msg */
    op_intrpt_schedule_self(op_sim_time() + LSAC_service_time, LSAC_MSG);
    break;

case T17_EXPIRE:
    /* find the timer that has expired and determine the link num*/
    link_num = dv_timer17[first_timer17].data;
    first_timer17 = mtp3_increment(first_timer17, 1, SYS_LINK_LIMIT);
    /* send start message to the link */
    if (SLM_link_table[link_num].active_status ==
    LINK_ACTIVATING_RESTORING_WAIT) {
        if (SLM_link_table[link_num].emergency) {
            mtp3_send_external_msg(SLM_link_table[link_num].link_objid,
            MSG_LSAC_LSC_EMERGENCY, 0, iciptr_external);
        }
        mtp3_send_external_msg(SLM_link_table[link_num].link_objid,
        MSG_LSAC_LSC_START, 0, iciptr_external);
        SLM_link_table[link_num].active_status = LINK_ACTIVATING_RESTORING;
    }
    break;
}
```

APPENDIX M. EXECUTIVES FOR “LSLA” STATE

EXECUTIVES FOR “LSLA” STATE

```
/* ====================================================================*/
/* Signaling Link Activation (LSLA), Signaling Link Management(SLM)     */
/* Signalling System 7 (SS7), Message Transfer Part Level 3 (MTP3)      */
/* Each message are processed according to the algorithm               */
/* as defined in ITU-T Q.704 (Jul 96), Fig 38.                          */
/* ====================================================================*/

/* Internal Message Interrupt                               */
/* If incoming message exists, remove the message packet */
/* from the buffer and process it                           */
/* The first field in each packet is always the message  */
/* code, it can be followed by one or more fields          */
/* depending on the message                                 */

if (!op_subq_empty(LSLA_BUFFER)) {
   pkptr = op_subq_pk_remove(LSLA_BUFFER, OPC_QPOS_HEAD);
   op_pk_fd_get(pkptr, FD_INDEX_1ST, &msg_code);
   op_pk_fd_get(pkptr, FD_INDEX_2ND, &link_num);

   switch (msg_code) {

       case MSG_LSAC_LSLA_START_ACTIVATION:
          if (dv_LSLA_state == STATE_LSLA_IDLE) {

             /* Physical selection and connection of datalink to    */
             /* data terminal suppose to take place here            */
             /* This is not needed for simulation so instant reply */
             /* to LSAC to ahead and start using the link           */
             LSLA_LSAC_START_SIG_LINK(link_num);
          }
          break;

       case MSG_LSAC_LSLA_RESTART_ACTIVATION:
          if (dv_LSLA_state == STATE_LSLA_IDLE) {

             /* Physical selection and connection of Datalink for  */
             /* replacement suppose to take place here              */
             /* This is not needed for simulation so instant reply */
             /* to LSAC that replacement is unsuccessful, try       */
             /* starting the link again                              */

             LSLA_LSAC_ACTIVATION_UNSUCCESSFUL(link_num);
             LSLA_LSAC_START_SIG_LINK(link_num);
          }
          break;
```

```
            default:
               op_pk_destroy(pkptr);
               break;
      }
   }

   /* schedule interrupt to check and process next LSLA msg */
   op_intrpt_schedule_self(op_sim_time() + LSLA_service_time, LSLA_MSG);
```

APPENDIX N. EXECUTIVES FOR "LSLR" STATE

EXECUTIVES FOR "LSLR" STATE

```
/* ==================================================================*/
/* Signaling Link Restoration (LSLR), Signaling Link Management(SLM)    */
/* Signalling System 7 (SS7), Message Transfer Part Level 3 (MTP3)      */
/* Each message are processed according to the algorithm                */
/* as defined in ITU-T Q.704 (Jul 96), Fig 39.                          */
/* ==================================================================*/

/* from the buffer and process it                         */
/* The first field in each packet is always the message   */
/* code, it can be followed by one or more fields         */
/* depending on the message                               */


if (!op_subq_empty(LSLR_BUFFER)) {
   pkptr = op_subq_pk_remove(LSLR_BUFFER, OPC_QPOS_HEAD);
   op_pk_fd_get(pkptr, FD_INDEX_1ST, &msg_code);
   op_pk_fd_get(pkptr, FD_INDEX_2ND, &link_num);


   switch (msg_code) {


       case MSG_LSAC_LSLR_START_RESTORATION:
          if (dv_LSLR_state == STATE_LSLR_IDLE) {


             /* Physical selection and connection of datalink to    */
             /* data terminal suppose to take place here            */
             /* This is not needed for simulation so instant reply */
             /* to LSAC to ahead and start using the link           */
             LSLR_LSAC_START_SIG_LINK(link_num);
          }
          break;


       case MSG_LSAC_LSLR_RESTART_RESTORATION:
          if (dv_LSLR_state == STATE_LSLR_IDLE) {


             /* Physical selection and connection of Datalink for  */
             /* replacement suppose to take place here              */
             /* This is not needed for simulation so instant reply */
             /* to LSAC that replacement is unsuccessful, try       */
             /* starting the link again                             */


             LSLR_LSAC_RESTORATION_UNSUCCESSFUL(link_num);
             LSLR_LSAC_START_SIG_LINK(link_num);
          }
          break;
```

```
        default:
          op_pk_destroy(pkptr);
          break;
    }
  }

  /* schedule interrupt to check and process next LSLA msg */
  op_intrpt_schedule_self(op_sim_time() + LSLR_service_time, LSLR_MSG);
```

APPENDIX O. EXECUTIVES FOR "LSLD" STATE

EXECUTIVES FOR "LSLD" STATE

```
/* ==================================================================*/
/* Signaling Link Deactivation (LSLD), Signaling Link Management(SLM)   */
/* Signalling System 7 (SS7), Message Transfer Part Level 3 (MTP3)     */
/* Each message are processed according to the algorithm               */
/* as defined in ITU-T Q.704 (Jul 96), Fig 40.                         */
/* ==================================================================*/

/* from the buffer and process it                         */
/* The first field in each packet is always the message  */
/* code, it can be followed by one or more fields         */
/* depending on the message                               */


if (!op_subq_empty(LSLD_BUFFER)) {
   pkptr = op_subq_pk_remove(LSLD_BUFFER, OPC_QPOS_HEAD);
   op_pk_fd_get(pkptr, FD_INDEX_1ST, &msg_code);
   op_pk_fd_get(pkptr, FD_INDEX_2ND, &link_num);


   switch (msg_code) {


       case MSG_LSAC_LSLD_DEACTIVATE_LINK:
          if (dv_LSLD_state == STATE_LSLD_IDLE) {

             /* Disconnection of datalink suppose to occur here    */
             /* However this is not needed for simulation and      */
             /* all data links are assume pre-connected and stays */
             /* so they do not come available                      */
             /* So this processor actually does nothing            */

             dv_LSLD_state = STATE_LSLD_IDLE;
          }
          break;

       default:
          op_pk_destroy(pkptr);
          break;
   }
}


/* schedule interrupt to check and process next LSLD msg */
op_intrpt_schedule_self(op_sim_time() + LSLD_service_time, LSLD_MSG);
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX P. EXECUTIVES FOR "LSTA" STATE

EXECUTIVES FOR "LSTA" STATE

```
/*==========================================================================*/
/* SS7 MTP3, Signaling Link Management (SLM)                                */
/* Terminal Allocation Control (LSTA)                                       */
/* Note: This function is NOT implemented                                   */
/*==========================================================================*/

/* destroy all msg packets in the queue */
if (!op_subq_empty(LSTA_BUFFER)) {
   op_subq_flush(LSTA_BUFFER);
}

/* schedule interrupt to check and destroy msg packets */
op_intrpt_schedule_self(op_sim_time() + LSTA_service_time, LSTA_MSG);
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX Q. EXECUTIVES FOR "LSDA" STATE

EXECUTIVES FOR "LSDA" STATE

```
/*==========================================================================*/
/* SS7 MTP3, Signaling Link Management (SLM)                                */
/* Datalink Allocation Control (LSDA)                                       */
/* Note: This function is NOT implemented                                   */
/*==========================================================================*/

/* destroy all msg packets in the queue */
if (!op_subq_empty(LSDA_BUFFER)) {
   op_subq_flush(LSDA_BUFFER);
}

/* schedule interrupt to check and destroy msg packets */
op_intrpt_schedule_self(op_sim_time() + LSDA_service_time, LSDA_MSG);
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX R. EXECUTIVES FOR ISUP LEVEL

HEADER BLOCK

```
/* define field index for use in OPNET */
#define FD_INDEX_1ST    1
#define FD_INDEX_2ND    2
#define FD_INDEX_3RD    3
#define FD_INDEX_4TH    4
#define FD_INDEX_5TH    5
#define FD_INDEX_6TH    6
#define FD_INDEX_7TH    7
#define FD_INDEX_8TH    8
#define FD_INDEX_9TH    9
#define FD_INDEX_10TH   10


/* define size of SS7 packet fields */
#define FD_LEN_PDU      -1 /* size of PDU set to -1 for encap   */
#define FD_LEN_SF       8  /* size of Status Field              */
#define FD_LEN_SIO      8  /* size of Service Information Octet */
#define FD_LEN_SI       4  /* size of Service Indicator         */
#define FD_LEN_SSF      4  /* size of Sub-Service Field         */
#define FD_LEN_DPC      14 /* size of Destination Point Code    */
#define FD_LEN_OPC      14 /* size of originating Point Code    */
#define FD_LEN_SLS      4  /* size of signaling link selection  */
#define FD_LEN_H0H1     8  /* size of H0H1 field                */
#define FD_LEN_SPARE2   2  /* size of spare field with 2 bits   */
#define FD_LEN_LABEL    32 /* size of a standard label          */
#define FD_LEN_PC       14 /* size of a point code              */


#define SIO_SI_ISDN_UP          0x5


/* definition of SS7 MTP Level 3 Sub-service Field (SSF)   */
#define SIO_SSF_INTERNATIONAL   0x0
#define SIO_SSF_NATIONAL        0x8


#define TX_MSU 1
#define RECEIVE  ((op_intrpt_type() == OPC_INTRPT_STRM) && (op_intrpt_strm()
                  == 0))
#define TRANSMIT ((op_intrpt_type() == OPC_INTRPT_SELF) && (op_intrpt_code()
                  == TX_MSU))
```

STATE VARIABLES BLOCK

```
/* Number of MSU received */
int    \MSU_tx_count;

Stathandle  \MSU_tx_count_handle;
```

```
Objid \own_objid;

int    \own_PC;
int    \dest_PC;

int    \MSU_rx_count;

Stathandle  \MSU_rx_count_handle;

Log_Handle  \log_handle;

Stathandle  \total_delay_handle;

int    \sls;
```

TEMPORARY VARIABLES BLOCK

```
Packet *pkptr;
Packet * pkptr_rx;
int i;
int data;
int remainding_packet_len;
double start_time;
double total_delay;
```

"INIT" STATE

```
own_objid = op_id_self();
op_ima_obj_attr_get(own_objid, "own_PC", &own_PC);
op_ima_obj_attr_get(own_objid, "dest_PC", &dest_PC);
MSU_rx_count = 0;
MSU_tx_count = 0;
sls = 0;
MSU_rx_count_handle = op_stat_reg("MSU_rx_count", OPC_STAT_INDEX_NONE,
                                  OPC_STAT_LOCAL);
MSU_tx_count_handle = op_stat_reg("MSU_tx_count", OPC_STAT_INDEX_NONE,
                                  OPC_STAT_LOCAL);
total_delay_handle = op_stat_reg("total delay", OPC_STAT_INDEX_NONE,
                                  OPC_STAT_LOCAL);
op_intrpt_schedule_self (op_sim_time() + 9, TX_MSU);
log_handle = op_prg_log_handle_create(OpC_Log_Category_Configuration,
                                      "ISDN UP", "link start up", 1000);
```

"GET" STATE

```
pkptr_rx = op_pk_get(0);
op_pk_fd_get(pkptr_rx, FD_INDEX_6TH, &data);
op_pk_fd_get(pkptr_rx, FD_INDEX_7TH, &start_time);
total_delay = op_sim_time() - start_time;
op_pk_destroy(pkptr_rx);
MSU_rx_count = MSU_rx_count + 1;
op_stat_write(MSU_rx_count_handle, MSU_rx_count);
op_stat_write(total_delay_handle, total_delay);
op_prg_log_entry_write(log_handle, "ISDN rec rx_count: %d, data: %d",
MSU_rx_count, data);
```

"SEND" STATE

```
pkptr = op_pk_create(0);
MSU_tx_count = MSU_tx_count + 1;
op_pk_fd_set(pkptr, FD_INDEX_1ST, OPC_FIELD_TYPE_INTEGER, SIO_SI_ISDN_UP,
             FD_LEN_SI);
op_pk_fd_set(pkptr, FD_INDEX_2ND, OPC_FIELD_TYPE_INTEGER, SIO_SSF_NATIONAL,
             FD_LEN_SSF);
op_pk_fd_set(pkptr, FD_INDEX_3RD, OPC_FIELD_TYPE_INTEGER, dest_PC,
             FD_LEN_DPC);
op_pk_fd_set(pkptr, FD_INDEX_4TH, OPC_FIELD_TYPE_INTEGER, own_PC,
             FD_LEN_OPC);

/*  Loop to implement load sharing  */
if (sls >= 15) {
  sls = 0;
}
else {
  sls = sls + 1;
}

op_pk_fd_set(pkptr, FD_INDEX_5TH, OPC_FIELD_TYPE_INTEGER, sls, FD_LEN_SLS);
op_pk_fd_set(pkptr, FD_INDEX_6TH, OPC_FIELD_TYPE_INTEGER, MSU_tx_count, 16);
op_pk_fd_set(pkptr, FD_INDEX_7TH, OPC_FIELD_TYPE_DOUBLE, op_sim_time(), 32);
remainding_packet_len = op_dist_exponential(184);
op_pk_fd_set(pkptr, FD_INDEX_8TH, OPC_FIELD_TYPE_INTEGER, 0,
             remainding_packet_len);

op_pk_send(pkptr, 0);
op_intrpt_schedule_self (op_sim_time() + op_dist_exponential(0.1), TX_MSU);
op_stat_write(MSU_tx_count_handle, MSU_tx_count);
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Russell, T., *Signaling System #7*, Second Edition, McGraw-Hill, New York, 1998.

2. International Telecommunications Union Telecommunications (ITU-T) Recommendation Q.701, *Introduction to CCITT Signaling System No. 7*, March 1993.

3. Bhatnagar, P.K., *Engineering Networks for Synchronization, CCS7, and ISDN: Standards, Protocols, Planning, and Testing*, IEEE Telecommunication Handbook Series, 1997.

4. Lin, Y.B., "Signaling System Number 7," *IEEE Potentials*, Vol. 15, Issue 3, August-September 1996.

5. Modarressi, A.R., and Skoog, R.A., "An Overview of Signaling System No. 7," *IEEE Communications Magazine*, Vol.80, Issue 4, pp. 590-606, April 1992.

6. Black, U., *ISDN and SS7: Architectures for Digital Signaling Networks*, Prentice Hall, New Jersey, 1997.

7. International Telecommunications Union Telecommunications (ITU-T) Recommendation Q.703, *Functional Description of Message Transfer Part (MTP) of Signaling System No. 7*, July 1996.

8. OPNET, *Modeling*, Vol.'s 1-2, MIL3, Inc., 3400 International Drive NW, Washington D.C., 20008, 1996.

9. OPNET, *Modeler Training Manual*, MIL3, Inc., 3400 International Drive NW, Washington D.C., 20008, 1998.

10. Lim, C.T., *Simulation of Signaling System No. 7: Message Transfer Part 2*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 2000.

11. International Telecommunications Union Telecommunications (ITU-T) Recommendation Q.704, *Functional Description of Message Transfer Part (MTP) of Signaling System No. 7*, July 1996.

12. Schwartz, M., *Broadband Integrated Networks*, Prentice Hall, New York, 1996.

13. MicroLegend Telecom Systems Inc., "Internet Telephony Tutorial – Integrating Internet Protocol and Intelligent Networks," [http://www.td.unige.ch/mis/microlegend/what-it.htm], May 1998.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center..2
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library.. 2
Naval Postgraduate School
411 Dyer Rd.
Monterey, CA 93943-5101

3. Chairman, Code EC...1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5121

4. Prof. John McEachen, Code EC/Mj 1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5121

5. Prof. Murali Tummala, Code EC/Tu 1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5121

6. Head Librarian..1
Defence Technology Tower Library
1 Depot Rd., #02-01
Defence Technology Tower A
Singapore 109681
Singapore

7. Naval Engineering Logistics Office...1
P.O. Box 15467
Arlington, VA 22215-5000

8. Ms. Rosemary Wenchel...1
Naval Information Warfare Analysis Center
9800 Savage Road
Ft. Meade, MD 20755

9. Dr. William Semancik..1
Laboratory for Telecommunications Science
9800 Savage Road
Ft. Meade, MD 20755
ATTN: R56

10. Ow Kong Chung... 1
Blk 212, #08-253,
Bishan Street 23,
Singapore 570212
Singapore

11. Tan, Ah Tuan..1
Blk 296, #03-04,
Choa Chu Kang Ave 2,
Singapore 680296
Singapore

12. Head EINS-AvB, ALD...1
HQ RSAF
AFPN 8060
303 Gombak Drive, #01-31
Singapore 669645
Singapore